

**FELICIA IONESCU**



**Inginerie electronică**

**GRAFICA**

**în realitatea virtuală**

**Editura Tehnica**

III 262.723



## **Colecția Inginerie electronică**

**Coordonatorul colecției: prof. univ. dr. ing. Mircea BODEA**

Editura TEHNICĂ își propune să găzduiască în cadrul colecției *Inginerie electronică* cărți fundamentale, care și-au demonstrat perenitatea prin nivelul deosebit de competitiv al abordării didactice și științifice a subiectului, dar și prin contribuția la satisfacerea cererii în continuă creștere de proiectanți de circuite și de sisteme microelectronice analogice, digitale sau semnale mixte.

Ca urmare, proiectarea de circuite și sisteme microelectronice, prin implicațiile sale științifice și economice, este un subiect de o importanță majoră care se regăsește rezonant atât în zona de interes a instruirii universitare și postuniversitare, cât și în aceea a interesului profesional al celor implicați în realizarea de sisteme microelectronice, indiferent de faptul că sunt juniori, seniori sau veterani.

Data fiind „geometria variabilă“ a domeniului de inginerie electronică și dinamica sa deosebit de accentuată, subiectele cărților acestei colecții se vor concentra îndeosebi pe fundamente, îmbinând abordarea didactică cu aceea de tip ingineresc. În acest fel, seria va acoperi atât cerințele ingineriei incrementale, în care dezvoltarea se face prin contiguitate, cât și pe cele ale ingineriei fundamentale, în care caracteristica majoră a dezvoltării o constituie discontinuitatea, ruperea de ritm.

**În această colecție vor apărea:**

- Gheorghe Ștefan **Circuite integrate digitale**
- Corneliu Burileanu **Microprocesoare – arhitecturi CISC și RISC**



Prof. univ. Felicia IONESCU

✓ 233.332

# GRAFICA ÎN REALITATEA VIRTUALĂ



022064  
B.C.U. - IASI



EDITURA TEHNICĂ  
București, 2000



## Prefață

Realitatea virtuală este o tehnologie care a modificat și continuă să modifice modul de abordare a numeroase aplicații practice în industrie, medicină, instruire, artă și divertisment. Deși începuturile realității virtuale se pot situa la nivelul anilor 1965-1970, legate de cercetările lui Ivan Sutherland și de realizarea unor simulatoare de antrenament pentru piloți, cercetări sistematizate în domeniul realității virtuale au avut loc mai cu seamă în ultimii 10-15 ani.

Realitatea virtuală înglobează o mare varietate de tehnologii și echipamente (echipamente electronice, hidraulice, sisteme de calcul în timp real, proiectoare optice), care generează senzații adresate simțurilor umane (simțului vizual, auditiv, tactil). Dintre acestea, simțul vizual este cel mai intens folosit în realitatea virtuală, deoarece cele mai multe informații despre mediul înconjurător se obțin prin intermediul văzului. Imaginea vizuală este componenta esențială a realității virtuale, fiind aceea care permite utilizatorului să identifice mediul în care evoluează, să se orienteze și să acționeze și, de aceea, cele mai multe cercetări în domeniul realității virtuale au fost făcute în domeniul graficii și al generării imaginilor.

Grafica utilizată în realitatea virtuală permite generarea prin calcul a imaginilor, pornind de la modelele obiectelor tridimensionale care alcătuiesc scena virtuală. Două cerințe de performanță ale graficii sunt vitale în sistemele de realitate virtuală: realismul imaginilor și generarea acestora în timp real. Aceste cerințe implică atât aspecte software, de selectare a celor mai adecvați algoritmi de generare a imaginilor, cât și aspecte hardware, de realizare a unor echipamente performante, care să asigure realismul și viteza de generare a imaginilor în realitatea virtuală. Stații grafice multiprocesor și acceleratoare grafice care implementează în hardware algoritmi de generare a imaginilor sunt echipamente de bază în realitatea virtuală, iar progresele tehnologice realizate au permis utilizarea acestora într-un număr imens de aplicații de realitate virtuală, accesibile în momentul de față unor categorii variate de utilizatori.

Realitatea virtuală este doar una dintre aplicațiile graficii pe calculator, alături de altele cum sunt: realizarea interfețelor utilizator dezvoltate în numeroase programe utilitare și medii de programare, proiectarea asistată de calculator (CAD – *Computer Aided Design*), prezentările grafice interactive, vizualizarea datelor științifice, tehnologia multimedia.

În această lucrare se pune accentul pe conexiunea dintre aspectele teoretice ale graficii și implementarea programelor de generare a imaginii obiectelor și scenelor virtuale tridimensionale. În primele capitole sunt prezentate operațiile de bază în grafica pe calculator: modelarea obiectelor tridimensionale, transformări



geometrice în spațiu, sisteme de vizualizare, transformarea de rastru. În cea de-a doua parte a lucrării se reiau toate aceste aspecte ale generării imaginilor din perspectiva programării, folosind biblioteca grafică OpenGL, sistemul de dezvoltare GLUT și limbajul de modelare în realitatea virtuală VRML. De asemenea, sunt prezentate aspectele avansate ale generării imaginilor tridimensionale (anti-aliasing, umbrire, texturare), atât din punct de vedere teoretic, cât și al abordării în programare.

Unul din scopuri principale ale cărții este de a oferi soluții de implementare a aplicațiilor grafice, în special cele necesare în realitatea virtuală, prin prezentarea metodelor actuale și accesibile de rezolvare a fiecărei etape a procesului de generare a imaginilor. În acest sens, aproape toate imaginile ilustrative ale diferitelor tehnici au fost produse de autoare prin programe grafice descrise în text. Cea mai mare parte a exemplelor prezentate se bazează pe experiența de implementare a unor sisteme grafice integrate în simulatoare de zbor, realizate în colaborare cu cercetătorii și inginerii de la Institutul de Simulatoare Simultec SA, București. De asemenea, materialul prezentat are ca suport experiența de curs *Grafică și Realitate Virtuală*, predat studenților anului IV de la Facultatea de Electronică și Telecomunicații, Universitatea Politehnica București.

Pentru realizarea lucrării am avut sprijinul permanent al mai multor colegi și al colectivului Laboratorului de Ingineria Informației din Catedra de Electronică Aplicată și Ingineria Informației. Regretatul prof. dr. ing. Adrian-Traian Murgan a fost cel care a introdus disciplina *Grafică în Realitatea Virtuală* la secția de Ingineria Informației din Facultatea de Electronică și Telecomunicații și a sprijinit activitatea în acest domeniu. Preparatorii Andrei Jalbă și Bogdan Popescu au realizat o parte importantă din implementările algoritmilor de generare a imaginilor.

Cartea se adresează în primul rând studenților din domeniul sistemelor de calcul și ingineria informației, fiind totodată utilă și inginerilor și proiectanților de aplicații grafice în realitatea virtuală sau în alte domenii. Indexul de termeni prezintă pentru fiecare denumire versiunea în limba engleză, ca bază comună de interpretare a termenilor, sau ca singura denumire pentru acei termeni a căror traducere este prea puțin intuitivă.

În domeniul graficii în realitatea virtuală, dezvoltările tehnologice sunt deosebit de dinamice: apar în permanență echipamente din ce în ce mai performante, la prețuri din ce în ce mai scăzute; apar numeroase biblioteci, limbaje și programe utilitare (toolkits) care propun diferite modalități de abordare a aplicațiilor grafice. Din acest motiv, în lucrare s-a insistat mai mult pe aspectele teoretice ale graficii și realității virtuale, și mai puțin pe descrieri tehnologice care pot fi depășite într-un timp foarte scurt.

Aprilie, 2000

Autoarea



# CUPRINS

<b>1</b>	<b>Introducere în realitatea virtuală</b>	<b>1</b>
1.1	Sisteme de realitate virtuală	3
1.2	Scurt istoric al realității virtuale	4
1.3	Componentele sistemelor de realitate virtuală	7
1.3.1	Captarea poziției utilizatorului	8
1.3.2	Generarea imaginii vizuale	10
1.3.3	Generarea sunetului	18
1.3.4	Simularea senzației tactile și a forței de reacție	19
1.4	Sisteme de referință tridimensionale	20
1.5	Reprezentarea culorilor în sistemele grafice	21
1.5.1	Modelul RGB	21
1.5.2	Modelul HSV	22
<b>2</b>	<b>Modelarea obiectelor</b>	<b>25</b>
2.1	Modelarea poligonală a obiectelor	26
2.1.1	Reprezentarea poligoanelor	26
2.1.2	Reprezentarea poliedrelor	29
2.1.3	Implementarea modelului poligonal	31
2.1.4	Generarea modelului din descrierea matematică	34
2.1.5	Generarea modelului prin baleiere spațială	35
2.1.6	Generarea modelului pornind de la o mulțime de puncte care aparțin suprafeței de frontieră a obiectului	36
2.1.7	Triangularizarea unei mulțimi de puncte	37
2.1.8	Redarea imaginii obiectelor poligonale	44
2.2	Modelarea obiectelor prin rețele de petice parametrice bicubice	45
2.3	Modelarea prin compunerea obiectelor	46
2.4	Modelarea prin divizare spațială	48
<b>3</b>	<b>Transformări geometrice</b>	<b>50</b>
3.1	Transformări geometrice în spațiu	51
3.1.1	Transformări geometrice primitive	51
3.1.2	Sistemul de coordonate omogene	54



3.1.3	Compunerea transformărilor geometrice	57
3.1.4	Transformări inverse	61
3.1.5	Transformarea sistemelor de referință	63
3.1.6	Alte transformări geometrice în spațiu	73
3.2	Transformări geometrice în plan	74
<b>4</b>	<b>Sisteme de vizualizare</b>	<b>75</b>
4.1	Transformarea de observare	77
4.2	Transformarea de proiecție	82
4.2.1	Proiecția paralelă	83
4.2.2	Proiecția perspectivă	83
4.2.3	Sistemul de referință normalizat	85
4.3	Sistemul de vizualizare standard	92
4.3.1	Definirea sistemului de referință de observare	93
4.3.2	Definirea transformării de normalizare	93
4.4	Sistemul de referință ecran 3D	97
4.5	Decuparea obiectelor	100
4.5.1	Decuparea în plan	100
4.5.2	Decuparea suprafețelor relativ la volumul de vizualizare	105
4.5.3	Eliminarea obiectelor aflate în exteriorul volumului de vizualizare	108
4.5.4	Volumul de delimitare	109
4.5.5	Detecția coliziunii	110
<b>5</b>	<b>Transformarea de rastru</b>	<b>111</b>
5.1	Generarea segmentelor de dreaptă	112
5.2	Generarea poligoanelor	114
5.3	Eliminarea suprafețelor ascunse	116
5.3.1	Compararea adâncimilor	117
5.3.2	Eliminarea suprafețelor ascunse în spațiul obiect	119
5.3.3	Eliminarea suprafețelor ascunse în spațiul imagine: algoritmul Z-buffer	119
5.3.4	Eliminarea suprafețelor orientate invers	123
5.3.5	Ascunderea suprafețelor coplanare	125
<b>6</b>	<b>Biblioteca grafică OpenGL</b>	<b>127</b>
6.1	Dezvoltarea programelor grafice folosind utilitarul GLUT	129
6.1.1	Funcții de control al ferestrei de afișare	129
6.1.2	Funcții callback	131
6.1.3	Generarea obiectelor tridimensionale	132
6.2	Caracteristicile bibliotecii OpenGL	132
6.2.1	Poarta de afișare OpenGL	133



6.2.2	Bufferul de cadru	133
6.2.3	Operațiile de bază OpenGL	135
6.2.4	Primitive geometrice	138
6.2.5	Reprezentarea culorilor în OpenGL	141
6.3	Sistemul de vizualizare OpenGL	143
6.3.1	Sistemele de referință	143
6.3.2	Transformări geometrice	145
6.4	Decuparea obiectelor în OpenGL	159
6.4.1	Eliminarea suprafețelor ascunse	160
6.4.2	Selecția suprafețelor în funcție de orientare	160
6.5	Liste de display OpenGL	161
<b>7</b>	<b>Modele de reflexie și iluminare</b>	<b>163</b>
7.1	Considerații teoretice asupra reflexiei luminii	164
7.2	Modelul de reflexie Phong	166
7.3	Modele de umbrire	169
7.3.1	Modelul de umbrire constantă	169
7.3.2	Modelul de umbrire Gouraud	169
7.3.3	Modelul de umbrire Phong	172
7.4	Generarea fenomenelor naturale	173
7.5	Funcțiile OpenGL de calcul al iluminării	175
7.5.1	Definirea surselor de lumină	175
7.5.2	Definirea proprietăților materialelor	177
7.5.3	Normalele în vârfurile primitivelor geometrice	182
7.5.4	Controlul poziției și al direcției surselor de lumină	184
7.5.5	Combinarea culorilor	189
<b>8</b>	<b>Modelarea și redarea suprafețelor parametrice</b>	<b>195</b>
8.1	Curbe Bézier	195
8.2	Curbe B-spline	197
8.2.1	Curbe B-spline uniforme	198
8.2.2	Curbe B-spline neuniforme	200
8.3	Suprafețe Bézier	202
8.4	Suprafețe B-spline	205
8.5	Extensia controlului parametric: NURBS și $\beta$ -spline	206
8.6	Redarea suprafețelor parametrice	206
8.7	Funcții OpenGL pentru redarea suprafețelor parametrice	208
8.7.1	Generarea și redarea curbelor Bézier	208
8.7.2	Generarea și redarea suprafețelor Bézier	211
8.7.3	Generarea și redarea curbelor și a suprafețelor B-spline	213
<b>9</b>	<b>Anti-aliasing</b>	<b>217</b>
9.1	Considerații teoretice asupra aliasing-ului	217



9.1.1	Tehnica de prefiltrare a imaginilor	221
9.1.2	Tehnica de supraeșantionare a imaginilor	225
9.1.3	Eșantionarea stocastică	227
9.2	Funcții OpenGL pentru redare anti-aliasing	228
9.2.1	Anti-aliasing prin combinarea culorilor	228
9.2.2	Anti-aliasing prin acumulare	231
<b>10</b>	<b>Texturarea</b>	<b>235</b>
10.1	Aplicația texturilor	235
10.1.1	Aplicația texturilor bidimensionale	236
10.1.1	Aplicația texturilor tridimensionale	241
10.2	Tehnici de anti-aliasing în texturare	241
10.3	Funcții OpenGL de texturare	245
10.3.1	Definirea texturilor	246
10.3.2	Stiva matricelor de texturare	253
10.3.3	Filtrarea texturilor	255
10.3.4	Generarea coordonatelor de texturare	260
<b>11</b>	<b>Aplicații ale realității virtuale</b>	<b>263</b>
11.1	Baze de date grafice	264
11.2	Crearea și redarea scenelor virtuale	265
11.2.1	Crearea grafului scenei	268
11.2.2	Redarea scenelor virtuale	280
11.3	Limbajul VRML	289
11.3.1	Specificațiile limbajului VRML	290
11.3.2	Construirea scenelor virtuale în VRML	293
11.3.3	Animația în VRML	299
11.3.3	Care este viitorul limbajului VRML?	300
11.4	Aplicații ale realității virtuale	300
11.4.1	Simulatoare de antrenament	300
11.4.2	Aplicații în medicină	304
11.4.3	Aplicații în artă și divertisment	306
11.4.4	Factorul uman, etic și estetic în realitatea virtuală	306
<b>Bibliografie</b>		<b>309</b>
<b>Index bilingv de termeni</b>		<b>313</b>
<b>Planșe</b>		<b>319</b>



## INTRODUCERE ÎN REALITATEA VIRTUALĂ

*Realitatea Virtuală* a cunoscut în ultimii ani o imensă dezvoltare și publicitate. În reviste, magazine, la televiziune, s-a prezentat această "nouă și fantastică tehnologie" din cele mai variate puncte de vedere. Chiar și definițiile date termenului *Realitate Virtuală* diferă de la un autor la altul, una din definițiile cele mai răspândite fiind următoarea: *un sistem de Realitate Virtuală este un sistem care creează unui utilizator impresia că este "imersat" (prezent) într-un mediu sintetic* [Gig93]. Termenul *Realitate Virtuală* (*Virtual Reality*) este unul dintre termenii folosiți pentru a defini astfel de experimente, dar se mai folosesc și termenii: *lume virtuală* (*Virtual World*), *mediu virtual* (*Virtual Environment*), *mediu sintetic* (*Synthetic Environment*), *realitate artificială* (*Artificial Reality*). Deși aceste denumiri sunt echivalente, unii specialiști le preferă, deoarece termenul de *Realitate Virtuală*, a fost și este suprautilizat și, de multe ori, asociat cu așteptări lipsite de realism. Termenul *Cyberspace*, care se mai folosește pentru a desemna un mediu virtual, este, oarecum, nepotrivit, deoarece este preluat din denumirea folosită de William Gibson în romanul de ficțiune "*The Matrix*", pentru descrierea unei lumi a viitorului, tehnocriminală, invadată de droguri, implantări cerebrale și furturi prin calculator. Este normal ca specialiști care muncesc din greu pentru progrese în domeniul realității virtuale să nu dorească asocierea acestui domeniu cu previziuni atât de sumbre.

Cuvântul "virtual" este folosit în mod obișnuit în domeniul calculatoarelor pentru a desemna o entitate care simulează o altă entitate. De exemplu, termenul "memorie virtuală" se referă la simularea memoriei principale prin memoria hard-discului. Sistemul de operare stochează pagini de memorie în fișiere pe disc, astfel că sistemul pare a avea o memorie mult mai mare decât memoria reală. Cuvântul "realitate" se referă la mediul perceput de om prin intermediul simțurilor. Deoarece "realitatea" este percepută prin intermediul simțurilor, este posibilă "simularea" acesteia prin furnizarea datelor percepute de unul sau mai multe simțuri. De aceea, *Realitatea Virtuală* se referă la modalitatea prin care calculatorul modifică modul în care o persoană percepe realitatea, prin simularea unei alte realități. Această realitate, sau mediu, simulată de calculator este numită *Realitate Virtuală*.



Alte definiții ale termenului *Realitate Virtuală*:

“Un sistem de *Realitate Virtuală* este o interfață care implică simularea în timp real și interacțiunea prin intermediul mai multor canale senzoriale. Aceste canale sunt simțurile omului: văzul, auzul, simțul tactil, mirosul și gustul” [Burd93].

“*Realitate Virtuală* este un sistem folosit pentru a crea o lume artificială pentru un utilizator astfel încât să aibă impresia că se află în această realitate în care se poate mișca și interacționa cu obiectele înconjurătoare” [Man95].

“*Realitatea Virtuală* permite explorarea unei lumi generate de calculator, fiind parte din acea lume artificială” [Sher92].

“*Realitatea Virtuală* este iluzia participanților de a participa într-un mediu sintetic, și nu observarea externă a acelui mediu” [Gig93].

Dintre cele cinci simțuri care sunt folosite pentru percepția realității, nu toate sunt la fel de importante în crearea unui mediu virtual. Simțul gustului și al mirosului au acțiuni limitate în perceperea realității (cu excepția servirii mesei!) și puține cercetări au fost efectuate pentru folosirea lor în medii virtuale.

Simțul tactil este mult mai util, mai ales atunci când se manipulează obiecte în mediul virtual. Greutatea, temperatura, duritatea, rezistența la efort, toate aceste informații se obțin prin simțul tactil. Din acest motiv, cercetări importante au fost făcute pentru a simula “atingerea” obiectelor virtuale și, în momentul de față, experimentele în mediul virtual permit generarea informațiilor tactile.

Cele mai importante simțuri folosite în realitatea virtuală sunt văzul și auzul, deoarece cele mai multe informații despre mediul înconjurător se obțin prin intermediul ochilor și al urechilor. Din acest motiv, cele mai multe cercetări în domeniul realității virtuale au fost făcute în domeniul generării imaginii și a sunetelor în medii virtuale. Chiar clasificarea sistemelor de realitate virtuală se bazează în principal pe modul de generare a imaginii mediului virtual.

Dintre aplicațiile cele mai importante ale realității virtuale se pot enumera:

- Simulatoare de antrenament, în special simulatoare de zbor, în care se pot exersa manevre dificile, fără a pune în pericol viața pilotului sau securitatea aparatului de zbor.
- Proiectare în diferite domenii de activitate (construcții, arhitectură). Proiectantul are posibilitatea să vadă rezultatele proiectului sub forma imaginii acestuia în timp real, să observe detaliile împreună cu alte persoane interesate, și să ia decizii de modificare înainte de construirea prototipului.
- Vizualizarea științifică, prin care se obține imaginea diferitelor modele sau fenomene inaccesibile observației directe (structuri atomice, fluxuri de informație, etc).
- În domeniul medical, în special chirurgie, se pot efectua experimente “la rece” de învățare a diferitelor proceduri, fără riscul vieții pacientului.
- Jocurile distractive și filmele de animație sunt unele din cele mai cunoscute aplicații de realitate virtuală.



## 1.1 SISTEME DE REALITATE VIRTUALĂ

Se pot distinge mai multe categorii de sisteme de realitate virtuală disponibile în momentul de față: sisteme de realitate virtuală imersive (*immersive VR*), sisteme de simulare (*simulation VR*), sisteme proiective (*projected VR*), teleprezență (*telepresence*), realitatea îmbogățită (*augmented reality*) și sisteme de realitate virtuală desktop (*desktop VR*) [Lou97], [Isd99].

Cea mai completă formă de realitate virtuală se obține în sistemele imersive. Într-un sistem de realitate virtuală imersiv, contactul participantului cu lumea reală este complet întrerupt, acestuia permițându-i-se să vadă numai imaginea mediului sintetic, să audă numai sunetele generate artificial și să interacționeze numai cu obiectele virtuale pe care le vede în scenă. Această incluziune totală a participantului în mediul virtual se obține prin dispozitive de afișare (display-uri) montate pe cap (*head-mounted display* – HMD), căști audio (*headphones*), mănușă de date (*data glove*) și îmbrăcăminte de date (*data suits*).

Cea mai cunoscută și utilizată formă de realitate virtuală este cea din sistemele de simulare (simulatoare). Într-un simulator, participantul este plasat într-o versiune aproape reală a unei cabine de vehicul (avion, elicopter, mașină, tren, navă maritimă, navă spațială). În această cabină participantul are posibilitatea de a interacționa cu comenzile de control reale ale vehiculului (manete, pedale, butoane, etc), în timp ce este creată imaginea mediului virtual în care se desfășoară experimentul, imagine care răspunde acțiunilor efectuate de participant. Acest tip de realitate virtuală, care se mai numește și semi-imersivă, își are originea în simulatoarele de zbor construite pentru antrenarea piloților.

În sistemele de realitate virtuală proiectivă, imaginea mediului tridimensional este proiectată pe unul sau mai multe ecrane, care pot fi văzute de unul sau mai mulți utilizatori. Imaginea afișată pe ecrane urmărește acțiunile unuia dintre utilizatori, care demonstrează anumite acțiuni sau concepte celorlalți utilizatori din grup.

În sistemele de teleprezență un operator uman este conectat prin intermediul unei interfețe la senzori de poziție și camere video plasate într-un mediu real. În astfel de sisteme, operatorul poate să observe acțiunile unui robot plasat într-un mediu inaccesibil (sau periculos) și să controleze mișcările acestuia de la o distanță sigură.

Sistemele de realitate îmbogățită combină informațiile generate de calculator cu cele ale unui mediu real [Azu97]. Spre deosebire de sistemele imersive, în care utilizatorul nu are nici un contact cu lumea reală, în realitatea îmbogățită utilizatorul percepe lumea reală, cu obiecte virtuale suprapuse peste imaginea acesteia. Aplicații ale realității îmbogățite se folosesc în medicină, planificarea mișcărilor roboților, aviația militară, jocuri distractive. În astfel de aplicații se folosește un dispozitiv de afișare montat pe cap (HMD) care suprapune date generate de calculator peste imaginea mediului real.

În sistemele de realitate virtuală desktop, imaginea vizuală a mediului virtual tridimensional este afișată pe monitorul unui calculator (în general PC).



Participantul interacționează cu mediul virtual prin dispozitive de intrare standard (tastatură, mouse, joystick). Aceste sisteme permit observarea mediului virtual printr-o "fereastră" (ecranul monitorului) și de aceea se mai numesc sisteme WoW (*Window On the World*). Sunt cele mai simple și mai ieftine sisteme de realitate virtuală, dar este de așteptat ca astfel de sisteme să cunoască în viitor dezvoltări spectaculoase, datorită apariției unui mare număr de acceleratoare grafice care permit redarea în timp real a unor imagini realiste.

În realitatea virtuală trebuie să fie captată poziția corpului uman, astfel încât imaginea redată de display să fie dirijată de mișcările utilizatorului în mediul virtual. De exemplu, întoarcerea capului utilizatorului este sesizată de dispozitivele de captare a poziției și imaginea sau sunetul redat reflectă această nouă poziție.

Termenul "imersiv" este folosit pentru a descrie un mediu virtual în care participantul (împreună cu mișcările corpului acestuia) fac parte integrantă din mediu. Există diferite grade de imersiune, depinzând de dispozitivele folosite, de hardware-ul și software-ul disponibil, de gradul de urmărire al mișcărilor utilizatorului. Un mediu virtual imersiv are următoarele caracteristici:

- Pentru cel puțin unul din canalele senzoriale importante, tipic cel vizual, toate datele recepționate de participant sunt cele generate de calculator și afișate pe display, fiind întreruptă complet recepționarea datelor corespunzătoare realității fizice.
- Sistemul de realitate virtuală conține modele (geometric, acustic, fizic, tactil), care sunt prelucrate pentru comanda dispozitivelor de redare.
- Imaginile afișate pe display prezintă mediul sintetic într-un mod consistent, cu posibilitatea de percepție a obiectelor în adâncime, cu mascare reciprocă - vizuală sau auditivă - corespunzătoare.
- În orice moment, datele afișate pe display corespund poziției participantului uman în mediul virtual.

Imersivitatea unui sistem de realitate virtuală nu este o caracteristică absolută, și gradul de imersivitate se poate aprecia într-un mod mai mult sau mai puțin obiectiv. De exemplu, un sistem dotat cu căști binaurale, date vizuale stereo și sistem de urmărire a poziției capului și a mâinii este "mai imersiv" decât un alt sistem care nu posedă sisteme de urmărire a poziției mâinii sau capului participantului. Imersivitatea este proprietatea sistemului de realitate virtuală care provoacă senzația de "prezență", adică participantul se simte prezent în "locul" din mediul virtual afișat pe display.

## 1.2 SCURT ISTORIC AL REALITĂȚII VIRTUALE

Realitatea virtuală, în forma în care este cunoscută în momentul de față, nu este o invenție de dată recentă, experimente care pot fi considerate ca aparținând acestui domeniu fiind datate cu mai mult de treizeci de ani în urmă.



Dintre acestea, invenția lui *Morton Heilig* realizată în anul 1962, numită *Sensorama*, este considerată primul sistem de realitate virtuală. Invenția lui Heilig avea toate caracteristicile unui sistem de realitate virtuală, mai puțin interactivitatea. Observatorul era "plimbat" prin New York pe o motocicletă, având posibilitatea de a simți vibrațiile motocicletei, curenții de aer, zgomote și mirosuri specifice și, bineînțeles, imaginea străzilor, prezentate ca film color pe ecranul unui televizor montat pe o cască. Ruta de parcurgere era preînregistrată și, deci, fixă. Heilig nu era inginer, ci specialist în cinematografie, și dorea să modifice experiența cinematografică clasică; în orice caz, imaginația lui Heilig a devansat timpul în care și-a desfășurat activitatea.

Experimentele lui Heilig au fost continuate de un mare specialist în grafica pe calculator, *Ivan Sutherland*, care a adus numeroase contribuții în acest domeniu. În 1968 Ivan Sutherland a descris un dispozitiv de afișare montat pe cap (*head-mounted display*- HMD), ce consta din două monitoare și un sistem optic cu oglinzi semitransparente, pe care se afișau imagini dependente de poziția capului. Acest dispozitiv permitea combinarea unei imagini sintetice stereoscopice cu imaginea reală a mediului, realizând astfel un sistem care în momentul de față este numit "realitate îmbogățită" (*augmented reality*). Sutherland a adus contribuții importante la generarea imaginilor vizuale prin intermediul calculatorului, proiectând unele dintre cele mai performante generatoare de imagini, realizate de compania *Evens&Sutherland*.

Un alt nume recunoscut pentru lucrările semnificative în domeniul realității virtuale este numele lui *Tom Furness*, care a activat în laboratoarele de cercetări medicale ale forțelor armate americane (*US Air Force's Armstrong Medical Research*). Împreună cu echipa sa, Furness a realizat în anul 1986 un simulator de zbor dotat cu o cabină sofisticată, iar pilotul era echipat cu un sistem de afișare montat pe cap, prin care putea obține diferite informații de zbor și tactică de luptă, prezentate grafic, suprapus peste imaginea mediului sintetic.

Un alt organism guvernamental din SUA interesat de dezvoltarea simulatoarelor se zbor a fost NASA (*National American Space Agency*), care avea nevoie de simulatoare pentru antrenarea astronautilor, dat fiind că antrenarea acestora în condițiile reale ale spațiului cosmic nu este posibilă. Un proiect semnificativ realizat de inginerii de la NASA este proiectul VIEW (*Virtual Interactive Environment Workstation*), care este un sistem multisenzorial de imersiune, incluzând un dispozitiv de afișare montat pe cap (HMD), căști auditive tridimensionale, sistem de recunoaștere a vorbirii, sistem de urmărire (*tracking*) a mișcării capului și mâinii, mănușă de date (*DataGlove*). În cadrul acestui proiect a fost experimentat un tip nou de mănușă de date. Jaron Lanier, unul dintre realizatorii acesteia, a devenit mai târziu directorul companiei VPL, una dintre primele companii care a deținut un patent pentru mănuși de date și a comercializat echipamente pentru sisteme de realitate virtuală.

Cele mai importante contribuții la dezvoltarea realității virtuale au fost prilejuite de realizarea simulatoarelor de antrenament, în special simulatoare de zbor în domeniul militar. Dar, cel puțin până în anii 80-85, majoritatea descoperirilor în domeniul echipamentelor simulatoarelor militare erau secrete și



nu se publicau. Situația s-a schimbat mai târziu, când diminuarea bugetelor militare a determinat direcționarea către aplicații civile ale realității virtuale, ca și posibilitatea de a fi cunoscute și utilizate echipamente complexe de navigare în mediul virtual.

Dezvoltarea simulatoarelor de zbor a fost aceea care a permis identificarea și înțelegerea profundă a cerințelor tehnice care se impun sistemelor de realitate virtuală, impunând ideea că sistemele de realitate virtuală (inclusiv simulatoarele) sunt eficiente numai dacă experiența este convingătoare din punct de vedere al participantului. Dintre aceste cerințe de performanță ale sistemelor de realitate virtuală se pot enumera:

- Viteză de actualizare a datelor afișate de cel puțin 30 cadre/secundă.
- Timp de întârziere trebuie să fie cât mai redus, astfel încât să nu fie perceptibilă diferența dintre momentul de execuție a unei acțiuni și răspunsul afișat pe display.
- Mediu virtual reprezentat trebuie să fie cât mai complex, cu număr mare de obiecte vizibile.
- Rezoluția de reprezentare a imaginilor trebuie să fie cât mai mare; includerea mai multor trăsături de realism al imaginilor: ascunderea reciprocă între obiecte, iluminare, umbrire, texturare.
- Simularea mișcării cabinei și a reacției (forța de răspuns) sistemelor de comandă.

Simulatoarele care prezintă toate aceste caracteristici la nivele acceptabile ating costuri de realizare foarte mari, de câteva milioane de dolari. Pentru simulatoarele de zbor acest cost poate fi acceptabil, dat fiind că economiile de antrenare în simulator a piloților, ca și diminuarea riscului de avariere a aparatelor de zbor în cursul antrenamentelor în condiții reale, sunt de asemenea considerabile. Pentru ca sisteme de realitate virtuală pentru alte aplicații să devină accesibile, trebuie ca performanțe similare celor arătate mai sus să fie obținute la prețuri mult mai mici. Acest deziderat se conturează totuși ca un fapt perfect posibil, datorită progreselor tehnologice în domeniile implicate în realitatea virtuală: echipamente de calcul de mare viteză, traductoare de poziție de mare precizie, display-uri stereografice color, recunoașterea și sinteza vorbirii, simularea simțului tactil.

În țara noastră au fost realizate mai multe simulatoare de antrenament, atât în domeniul civil cât și în domeniul militar. Primul simulator de antrenament a fost simulatorul de locomotivă Diesel electrică, dezvoltat la Institutul de Cercetări și Proiectări Tehnologice în Transporturi (ICPTT), în colaborare cu Institutul de Tehnică de Calcul (ITC) în anul 1980. Deși mijloacele de calcul disponibile în acea perioadă, mai ales în condițiile din țara noastră, erau destul de precare, simulatorul a permis antrenarea a numeroși mecanici de locomotivă, în special pentru rezolvarea diferitelor situații de avarie. În condiții similare au mai fost realizate simulatoare de radar și de navigație. În domeniul simulatoarelor de zbor pentru avioane și elicoptere militare, primele simulatoare pentru avionul de producție românească IAR 93 au fost realizate în anii 1984-1989 la INCREST, în colaborare cu ITC, pe baza unor sisteme multiprocesor special proiectate.



În anul 1995, simulatorul elicopterului PUMA, realizat la Institutul de Simulatoare (Simultec), începea, deja, să se apropie de performanțele cerute pentru astfel de sisteme. Calculatorul central este o stație Silicon Graphics Indigo 2, iar ca generator de imagine s-a folosit un sistem multiprocesor Division, special conceput pentru aplicații de realitate virtuală. Cabina echipată asemănător unei cabine reale, sistemul de mișcare cu 6 grade de libertate, simularea eforturilor în comenzi și a altor sisteme de navigație aeriană completează funcționalitatea unui simulator de zbor cu reale calități de antrenament al poliților.

În 1998 a fost realizat simulatorul pentru avionul Mig-21 Lancer, la nivelul realizărilor actuale în acest domeniu. Construit pe bază unei stații Silicon Graphics Onyx2 Infinite Reality, cu o bază de date grafice care acoperă teritoriul întregii țări, cu sistem de mișcare cu șase grade de libertate, simulatorul Mig-21 Lancer este o realizare importantă din țara noastră în domeniul realității virtuale.

### 1.3 COMPONENTELE SISTEMELOR DE REALITATE VIRTUALĂ

Un sistem de realitate virtuală este compus din mai multe subsisteme care comunică între ele pentru redarea interacțiunii între utilizator și mediul virtual. În fig. 1.1 este prezentată schema bloc a unui sistem de realitate virtuală.

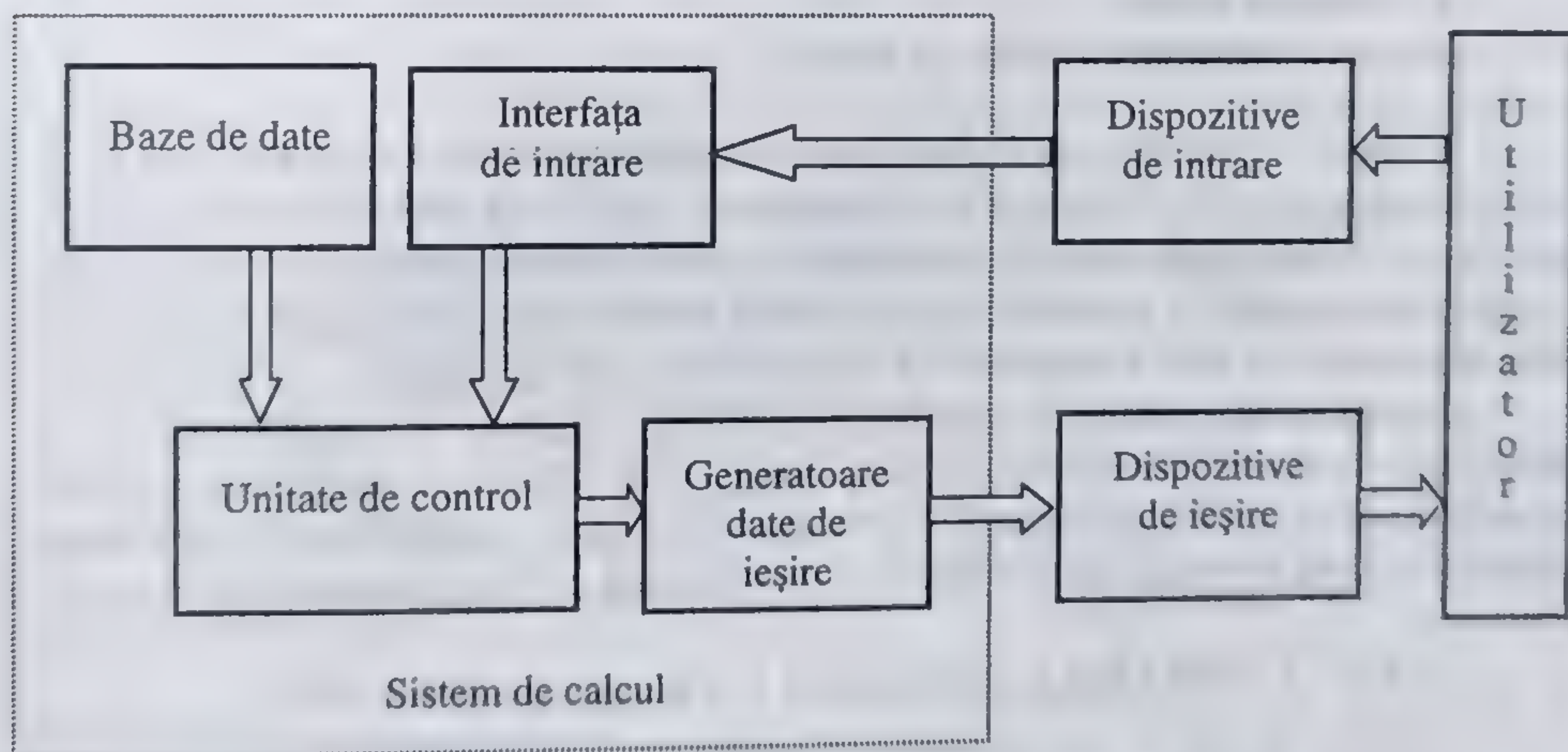


Fig. 1.1 Schema bloc a unui sistem de realitate virtuală.

Un sistem de realitate virtuală este compus dintr-un sistem de calcul, care prelucrează datele de modelare a mediului virtual (conținute în bazele de date) și generează imaginile corespunzătoare acțiunilor efectuate de utilizator. Sistemul de calcul implicat în realitatea virtuală este un sistem complex, fiind de cele mai multe ori un sistem paralel sau distribuit, cu interfețe specializate pentru recepționarea datelor de la dispozitivele de intrare și cu generatoare de imagini specializate.



Dispozitivele de intrare specifice realității virtuale sunt traductoare care convertesc acțiunile utilizatorului într-o formă interpretabilă de către calculator. Acțiunile sesizate și transmise sistemului sunt mișcările (ale capului, ale mâinii, ale corpului) și vorbirea. Dispozitivele de afișare specifice realității virtuale sunt display-uri video, căști audio, mănuși senzitive.

Se observă că un sistem de realitate virtuală este un sistem de control în buclă închisă, unul din elementele buclei fiind utilizatorul, deci este un sistem interactiv. În lucrarea sa [Burd94], G. Burdea definește sistemele de realitate virtuală prin trei "I": *imersivitate*, *interactivitate*, la care se adaugă *imaginația* necesară pentru ca mediul virtual creat să imite cât mai fidel realitatea fizică modelată.

Cerințele de imersiune și interactivitate impun cerința de *sistem de timp real* pentru realitatea virtuală. Pentru ca utilizatorul să se simtă o parte a mediului virtual, trebuie ca sistemul să răspundă acțiunilor sale într-un interval de timp suficient de mic, astfel încât diferența dintre momentul de execuție a unei acțiuni și răspunsul recepționat să nu fie perceptibilă. Această cerință de execuție în timp real impune folosirea unor sisteme de calcul și dispozitive de intrare și de ieșire performante, și de aici provine cea mai mare parte din costurile necesare pentru crearea mediilor virtuale.

Pentru crearea unui mediu virtual se generează și se redau pe dispozitive corespunzătoare:

- imaginea vizuală
- sunetul virtual
- senzația tactilă și forța de reacție

Imaginea vizuală a mediului virtual se generează prin calcul pornind de la modelele obiectelor din mediul virtual memorate sub formă de baze de date (scena virtuală), corespunzător poziției și acțiunilor utilizatorului. Imaginea vizuală este componenta esențială a realității virtuale, fiind aceea care permite utilizatorului să identifice mediul în care evoluează, să se orienteze și să acționeze.

Completarea imaginii vizuale cu sunetul virtual și senzația tactilă a mediului este foarte importantă pentru majoritatea experimentelor de realitate virtuală. În mod similar, generarea sunetului virtual și a senzației tactile trebuie să asigure fidelitate și viteză de răspuns cât mai adecvate aplicației respective.

### 1.3.1 CAPTAREA POZIȚIEI UTILIZATORULUI

Mediul sintetic în care evoluează utilizatorul unui sistem de realitate virtuală (numit și *cybernaut*) este definit într-un sistem de coordonate cartezian tridimensional numit sistem de coordonate universal (*world coordinate system*). Fiecare obiect are coordonatele lui relativ la acest sistem de referință și, de asemenea, utilizatorul are o poziție și o orientare definită în acest sistem de referință. Toate obiectele pe care utilizatorul le poate vedea la un moment dat se află în interiorul unei piramide numite piramidă de vizualizare, care este o subdiviziune a spațiului determinată de poziția utilizatorului, orientarea capului



acestui și unghiul de vizibilitate (aceste noțiuni vor fi descrise pe larg în cap. 3). Obiectele aflate în afara acestui volum se află încă în mediul virtual, dar nu sunt vizibile din poziția și orientarea dată a utilizatorului.

Pentru redarea imaginii obiectelor din mediul virtual trebuie, așadar, să fie captată (urmărită) poziția și orientarea capului utilizatorului, care definește poziția și direcția de observare, poziția mâinii utilizatorului, care permite identificarea interacțiunilor cu mediul virtual și, uneori, poziția și a altor părți ale corpului, pentru redarea mișcării sau a expresiei feței.

În sistemele grafice convenționale sau în sistemele de realitate virtuală neimersive se poate urmări poziția utilizatorului folosind dispozitive ca mouse, trackball sau joystick, cu trei sau șase grade de libertate, disponibile comercial de la firme ca Logitech sau Mouse System Corp. Dispozitivele cu trei grade de libertate permit măsurarea coordonatelor spațiale  $x$ ,  $y$ ,  $z$ , iar cele cu șase grade de libertate măsoară în plus rotațiile după cele trei axe.

În realitatea virtuală imersivă este necesară corelarea comenzilor senzoriale cu poziția și acțiunile utilizatorului, pentru a asigura senzația de imersiune. Dacă se urmărește poziția capului utilizatorului, atunci când acesta întoarce capul într-o anumită direcție, imaginea afișată pe display corespunde acelei direcții, și senzația de imersiune a utilizatorului este foarte puternică. În plus, cunoscând poziția capului, se poate calcula corect imaginea stereoscopică, compusă din două imagini diferite, corespunzătoare pozițiilor diferite ale celor doi ochi, și sunetul virtual, corespunzător poziției celor două urechi.

În mod asemănător, este necesar să fie cunoscută poziția mâinii utilizatorului, pentru calculul atingerii obiectelor virtuale. De aceea, trebuie să fie captată poziția capului sau a mâinii, folosind dispozitive de captare montate pe casca de vizualizare, pe căștile audio sau pe mănușa senzitivă.

Un dispozitiv de captare și urmărire a poziției utilizatorului (*tracker*) transmite datele de poziție și orientare (a capului sau a mâinii) către sistemul de generare a imaginilor, care le folosește pentru calculul și afișarea obiectelor corespunzătoare și a interacțiunii cu acestea.

Indiferent de tehnologia de realizare, dispozitivele de captare se caracterizează prin mai mulți parametri, și anume viteza de captare, timpul de răspuns al dispozitivului, precizia. Viteza de captare depinde de frecvența de eșantionare a măsurării și are valori tipice între 30 și 120 măsurări/secundă. Timpul de răspuns este durata de timp între modificarea poziției utilizatorului și momentul în care aceasta este recepționată în sistemul de calcul. Precizia este dată de diferența dintre valoarea reală a poziției și valoarea măsurată de dispozitivul de captare; cu cât această diferență este mai mică, cu atât precizia este mai bună. Dispozitivele de captare a poziției în sistemele de realitate virtuală trebuie să aibă precizie cât mai bună, viteză de captare cât mai mare și timp de răspuns cât mai mic. Dispozitivele care pot fi folosite pentru captarea poziției utilizatorului în spațiul tridimensional sunt de mai multe tipuri: magnetice, cu ultrasunete, optice.

Dispozitivele de captare magnetice sunt compuse dintr-un ansamblu emițător-receptor. Emițătorul este montat într-o poziție fixă și generează un câmp magnetic de joasă frecvență care este recepționat de antenele receptorului, atașat



utilizatorului aflat în mișcare. Din valoarea semnalului recepționat se poate determina poziția relativă a receptorului față de emițător. Pentru determinarea poziției capului, receptorul se montează pe cască de vizualizare; pentru determinarea poziției mâinii, receptorul se montează pe mânășă de date. Aceste dispozitive au ca dezavantaje un timp de răspuns ridicat și perturbarea câmpului magnetic de către obiectele metalice din raza de acțiune a emițătorului. Companii care produc dispozitive de captare magnetice sunt Polhemus și Ascension.

Dispozitivele de captare cu ultrasunete folosesc emițătoare și receptoare de ultrasunete pentru determinarea poziției în spațiu a utilizatorului. Emițătorul este compus din trei difuzoare ultrasonore și este montat într-o poziție fixă, iar receptorul conține trei microfoane și este montat pe cască sau pe mânășă de date a utilizatorului. La fel ca și în cazul dispozitivelor magnetice, semnalul captat de receptor permite determinarea poziției relative a receptorului față de emițător. Dispozitive de captare cu ultrasunete sunt produse de firmele Logitech și Transition State.

Dispozitivele de captare optice folosesc grile de diode electroluminescente (*Light Electroluminescent Diodes* – LED) dispuse în poziție fixă și o cameră video montată pe cască de utilizatorului. Diodele sunt activate în impulsuri, iar semnalul recepționat de cameră este prelucrat pentru identificare poziției relative a camerei față de grila de diode. Dezavantajul major al acestor dispozitive îl constituie dimensiunea limitată a grilei de diode, care limitează unghiul de rotație ce poate fi măsurat. Una din companiile care produc dispozitive de captare optice este Origin Instruments.

### 1.3.2 GENERAREA IMAGINII VIZUALE

Generarea imaginii vizuale în realitatea virtuală implică două aspecte importante: crearea modelului scenei virtuale și vizualizarea scenei virtuale. Crearea modelului scenei virtuale (mai pe scurt, crearea scenei virtuale) este un proces off-line și, de cele mai multe ori, de durată considerabilă, prin care se creează colecția de modele ale obiectelor tridimensionale care constituie cea mai adecvată reprezentare a mediului virtual. Vizualizarea scenei virtuale este un proces on-line, care se desfășoară în timp real, cu participarea uneia sau mai multor persoane, în care scena virtuală este explorată în mod interactiv și, în fiecare moment, imaginea scenei redată pe display depinde de condițiile de explorare (poziție de observare, acțiuni interactive, etc).

Frecvența de redare a imaginilor vizuale succesive (*update rate*) trebuie să fie mai mare decât frecvența limită a percepției vizuale a imaginilor distincte, care se situează în jurul a 15-20 imagini/secundă. Frecvența de redare a imaginii depinde de frecvența de refresh a display-ului folosit, fiind un submultiplu al acesteia. În mod obișnuit, în realitatea virtuală actualizarea imaginii afișate se face la un multiplu întreg de cadre baleiate. La frecvențe de refresh a display-ului de 60 cadre/secundă se generează imagini cu frecvența de 30 imagini/secundă sau 60 imagini/secundă.



Pentru crearea unei imagini vizuale cât mai apropiate de realitatea fizică a mediului modelat, este necesar ca scena virtuală să conțină cât mai multe obiecte, redată cât mai realist. Generarea imaginii unui număr mare de obiecte ale scenei virtuale într-un interval de timp impus necesită o putere de calcul considerabilă și de aceea generatoarele de imagine vizuală din sistemele de realitate virtuală sunt realizate pe baza unor arhitecturi paralele specializate care implementează accelerarea hardware a operațiile grafice.

Cel mai important indice de performanță al unui generator de imagine vizuală este *viteza de generare a imaginii*, specificată prin numărul de poligoane redată pe secundă. Acest mod de specificare rezultă din faptul că, în sistemele grafice utilizate în realitatea virtuală, obiectele tridimensionale sunt reprezentate în mod obișnuit printr-o colecție de poligoane.

Calitatea imaginii generate este caracterizată de mai mulți parametri, ca de exemplu rezoluția, modelul de umbrire, capacitatea de texturare, filtrarea anti-aliasing, etc. Rezoluția imaginii se exprimă prin numărul de elemente de imagine (pixeli) afișate pe display. Rezoluțiile la care se generează imaginea în sistemele de realitate virtuală variază de la  $640 \times 480$  pixeli până la valori de  $2048 \times 1280$  pixeli. Semnificația acestor parametri poate fi înțeleasă mai bine după parcurgerea capitolelor următoare; în această parte se vor aminti succint acești parametri pentru o idee de ansamblu asupra generatoarelor de imagine vizuale disponibile.

Pentru generarea imaginii în sistemele de realitate virtuală sunt necesare mai multe componente hardware (generatoare de imagine, dispozitive de afișare) și componente software. Aceste componente vor fi descrise în continuare.

### 1.3.2.1 Generatoare de imagine

Un generator de imagine vizuală poate fi un echipament separat, care se conectează la un calculator gazdă (host), sau poate fi o componentă a sistemului de calcul, numită subsistem grafic sau accelerator grafic. Stațiile grafice folosite în sistemele de realitate virtuală conțin subsisteme grafice puternice pentru generarea în timp real a imaginii obiectelor tridimensionale.

În ultimii ani au fost construite numeroase generatoare de imagine cu performanțe ridicate pentru aplicații de realitate virtuală, la prețuri de la câteva zeci de mii de dolari la câteva sute de mii de dolari. Performanțele și prețurile acestora au o dinamică extraordinară de mare, astfel încât o prezentare făcută la un moment dat poate deveni depășită foarte curând. De aceea, vor fi prezentate succint doar câteva generatoare reprezentative, care să dea o idee a performanțelor ce se pot atinge în generarea imaginilor: ProVision (produs de compania Division), Evens&Sutherland, Thomson CSF, stațiile grafice Silicon Graphics.

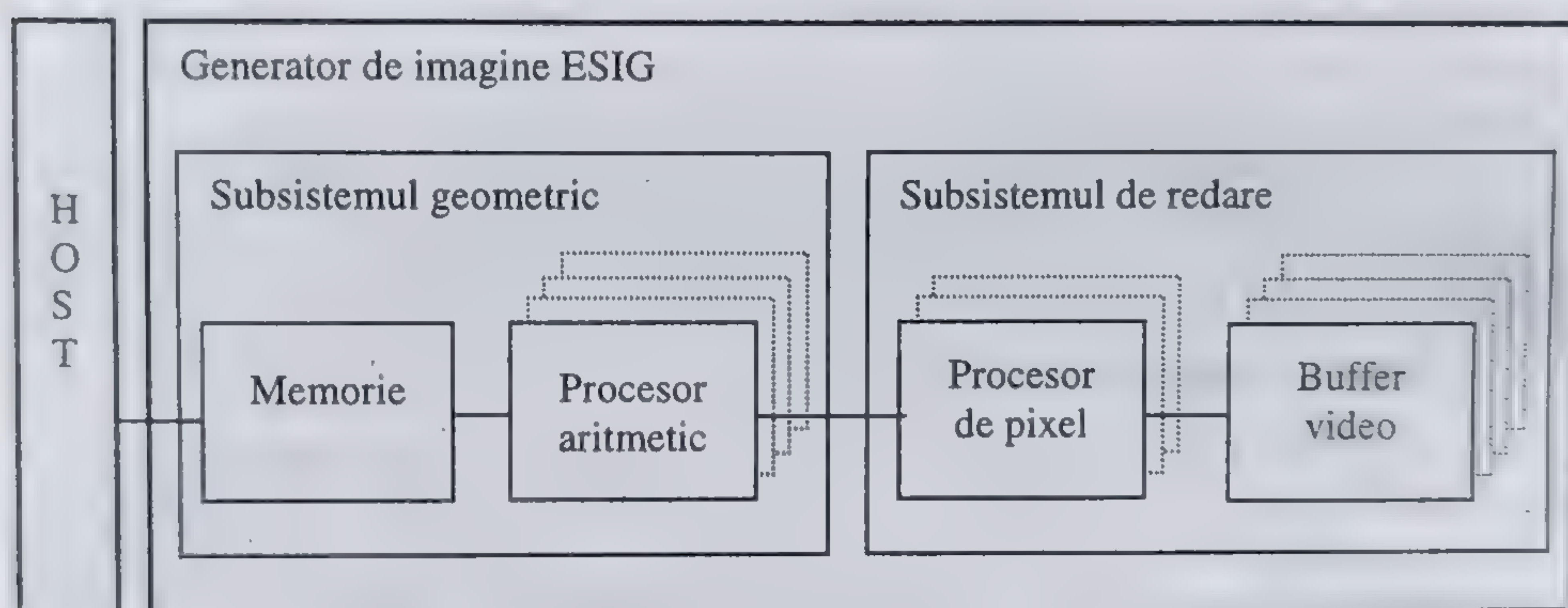
**Generatorul de imagine vizuală ProVision**, produs de firma Division în anii 1992-1996, a fost realizat într-o arhitectură paralelă, compus din 10 procesoare de uz general (3 procesoare Intel i860, 2 transputere Inmos T805 și 5 transputere Inmos 425) pentru calcule geometrice și o matrice de  $64 \times 128 = 8192$  de procesoare specializate care efectuează calculele de redare grafică cu umbrire și texturare [Grim93].



Un astfel de generator de imagine are o performanță de 30 000 poligoane/secundă cu umbrire Gouraud, la o rezoluție de maximum  $1024 \times 1024$  pixeli. Mai multe astfel de generatoare conectate la un calculator gazdă (host), care poate fi o stație Sun sau un calculator PC, permit generarea imaginii pe mai multe ecrane (canale de imagine), obținându-se astfel unghiuri de vizibilitate mari.

**Generatoarele de imagine Evens&Sutherland** au fost realizate în mai multe variante cu performanțe ridicate: ESIG-2000, ESIG-3000, ESIG-4000. Un generator de imagine ESIG (*Evens-Sutherland Image Generator*) este realizat într-o arhitectură paralelă constând din procesoare specializate (procesoare aritmetice și procesoare de prelucrare a pixelilor), conectate la un calculator host (fig. 1.2).

Viteza de generare a imaginii a unui generator ESIG 4000 este de  $9 \times 10^6$  de poligoane/secundă la o rezoluție de  $2048 \times 1280$ , la care se adaugă numeroase caracteristici de realism a imaginii redată: texturare cu interpolare trilineară, anti-aliasing, Z-buffer, simularea ceții, etc. Bineînțeles, prețurile sunt destul de ridicate (câteva sute de mii de dolari) și aceste generatoare sunt utilizate în special în simulatoarele de zbor.



**Fig. 1.2** Arhitectura generatoarelor de imagine ESIG.

Din aceeași categorie de generatoare de imagine cu performanțe și prețuri ridicate, folosite în special în simulatoarele de zbor, se mai pot aminti: Real 3D, produs de firma Lockheed Martin; P10 și Barracuda, produse de firma Primary Image; Ivex Visual System, produs de firma Ivex; Maxvue, produs de firma CAE Electronics; Star G2000, produs de firma Star Technology; Apogée, produs de firma Sogitec; VISA, produs de firma Thomson-CSF.

**Stațiile Silicon Graphics.** O soluție frecventă pentru generarea imaginii în realitatea virtuală este folosirea supercalculatoarelor de vizualizare, cum sunt stațiile Silicon Graphics [SGI96]. Combinația dintre unitatea centrală în arhitectură multiprocesor puternic și scalabil și un subsistem grafic, de asemenea scalabil, permite selectarea unei variante optime ca preț și performanțe pentru o aplicație dată. În ultimul deceniu au fost produse o gamă întreagă de variante ale stațiilor



Silicon Graphics: Indigo2, Indy, Octane, O2, Onyx2, care au prețuri de la câteva zeci de mii de dolari până la câteva sute de mii de dolari, în funcție de numărul de procesoare și subsistemul grafic inclus. Stația Onyx2 reprezintă una din soluțiile cele mai puternice de realizare a sistemelor de realitate virtuală. Pe lângă capacitatea de redare a imaginii tridimensionale, stația Onyx2 integrează numeroase alte interfețe și dispozitive de creare a mediului virtual: captarea poziției utilizatorului, mânășă de date, dispozitiv HMD, sistem de proiecție.

Arhitectura stației Onyx2 constă dintr-un număr de maximum 12 noduri de procesare, conectate între ele printr-o rețea de interconectare hipercub. Fiecare nod conține unul sau două procesoare și 4 GB de memorie, deci sistemul admite maximum 24 de procesoare R10000 (produse de firma MIPS) și 48 GB de memorie. La fiecare stație se pot conecta unul sau mai multe (maximum patru) subsisteme grafice pentru prelucrarea și generarea imaginii cu performanțe ridicate. Subsistemele grafice (ultimele versiuni, InfiniteReality și RealityMonster) sunt sisteme paralele specializate, care implementează hardware operații geometrice și de redare a primitivelor grafice cu trăsături avansate de realism, cum ar fi texturarea, iluminarea, umbrirea, anti-aliasing. Subsistemul grafic InfiniteReality conține un pipeline grafic, compus dintr-o succesiune de trei subsisteme (fig. 1.3):

- Subsistemul geometric (*Geometry Engine* – GE)
- Subsistemul de rastru (*Raster Manager* – RM)
- Subsistemul de afișare (*Display Generator* – DG)

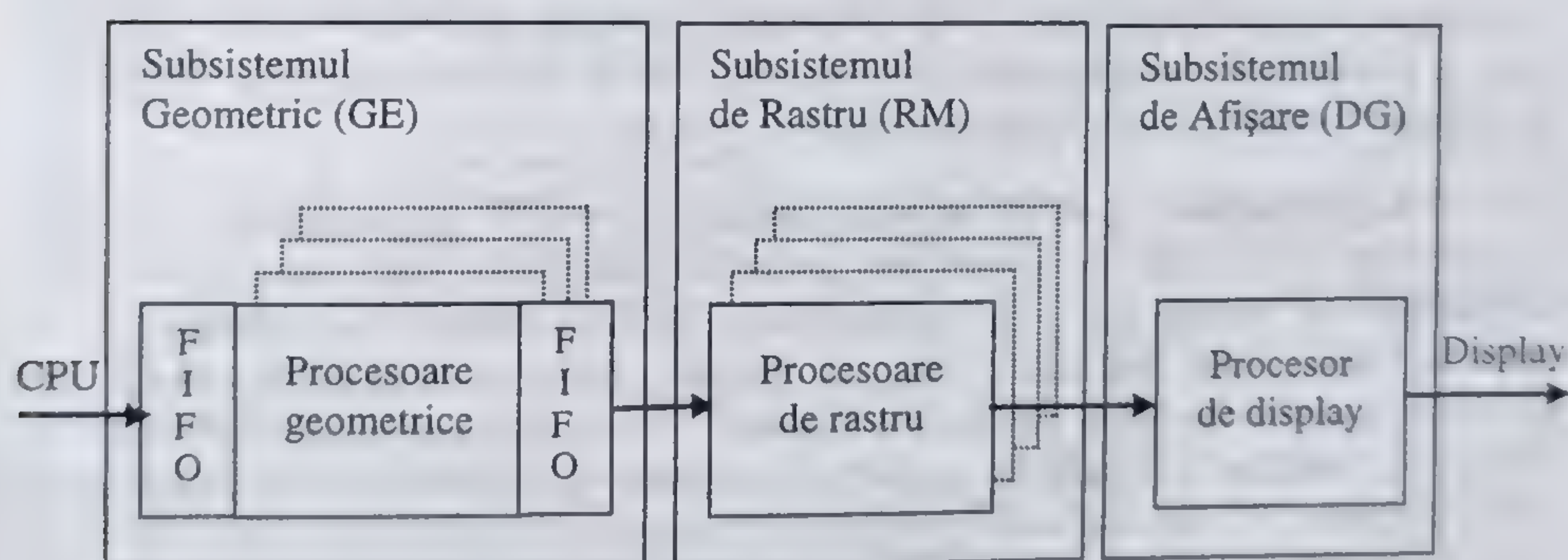


Fig. 1.3 Arhitectura subsistemului grafic InfiniteReality.

*Subsistemul geometric* execută primul stadiu de procesare grafică pipeline asupra datelor (transformări geometrice, calcule de iluminare, convoluții, histograme) prin intermediul unor acceleratoare hardware dedicate. Subsistemul geometric este, la rândul lui, un multiprocesor, compus din procesoare specializate numite mașini geometrice (*Geometry Engine* – GE), conectate între ele. Conectarea subsistemului geometric cu celelalte subsisteme este realizată prin intermediul unor buffere FIFO de dimensiuni mari, ceea ce asigură posibilitatea ca acesta să poată



execuția continuă, fără întreruperi, indiferent de starea celorlalte segmente ale pipeline-ului.

*Subsistemul de rastru* constă din una sau mai multe plăci de tip *Raster Manager* (RM), compuse din procesoare grafice specializate (procesoare de rastru) și buffere de imagine. Un subsistem de rastru poate conține una, două sau patru plăci RM, care asigură reprezentarea a 2,62 Mpixel (milioane de pixeli), respectiv 5,24 Mpixel și 10,48 Mpixeli. Modulele RM primesc de la subsistemul geometric date care descriu primitive grafice (puncte, linii, triunghiuri), pe care le transformă în imagine de pixeli în bufferul de imagine (*frame-buffer*). Prin utilizarea extensivă a paralelismului, subsistemul de rastru asigură prelucrarea anti-aliasing, a texturilor și a efectelor atmosferice, la nivel de pixel în timp real.

*Subsistemul de afișare* preia imaginile memorate în bufferul de imagine de către subsistemul de rastru și procesează pixelii prin convertire digital-analogă, pentru a produce fluxul de valori analogice a pixelilor, adecvat afișării pe video-monitoare sau video-proiectoare RGB de înalta rezoluție. Subsistemul de afișare permite programarea formatelor de afișare, pentru o varietate de monitoare, rezoluții, frecvențe de refresh și caracteristici de întreținere.

Un pipeline grafic InfiniteReality cu 4 procesoare geometrice GE, o placă de controler de rastru RM și un subsistem de afișare DG programat pentru o configurație cu rezoluție  $1280 \times 1024$ , 60Hz neîntreșut, are performanțele grafice de vârf de  $11 \times 10^6$  poligoane/secundă, cu texturare, anti-aliasing, Z-buffer.

Suportul software al stațiilor Silicon Graphics, incluzând sistem de operare multiprocesor derivat din Unix (IRIXx 6.x), sistem de timp real REACT, interfața de programare grafică de nivel înalt Performer, interfața grafică standard OpenGL, precum și multe alte instrumente software, permite dezvoltarea rapidă a aplicațiilor de realitate virtuală cu performanțe ridicate.

**Acceleratoare grafice pentru calculatoare PC.** În momentul de față, concomitent cu scăderea prețurilor supercalculatoarelor de vizualizare sunt disponibile un număr mare de acceleratoare grafice pentru calculatoare PC în variante mono sau multiprocesor, la prețuri mult mai scăzute decât ale generatoarelor de imagine prezentate anterior. Acest lucru permite dezvoltări de sisteme de realitate virtuală în numeroase domenii de activitate. Acceleratoarele grafice pentru PC se bazează pe circuite integrate specializate (chipset) care implementează în hardware o parte sau toate operațiile grafice, ceea ce le conferă diferite capacități de viteză și realism a imaginii generate. S-au proiectat și fabricat mai multe tipuri de circuite grafice specializate, pe baza cărora s-au realizat acceleratoare grafice 3D.

În anul 1996 firma Evans&Sutherland, specializată în generatoare de imagine performante, a realizat chipul Reallmage, care accelerează hardware toate primitivele interfeței grafice OpenGL (această interfață este prezentată în capitolul 6). Chipurile Reallmage sunt folosite în echiparea plăcilor grafice accelerate 3D AccelGalaxy și E&S Lightning 1200 pentru calculatoare PC sub sistemul de operare Windows NT 4.0 și au o viteză de generare de aproximativ  $3 \times 10^6$  triunghiuri/secundă.



Alte chipset-uri folosite în realizarea acceleratoarelor grafice sunt GLINT de la firma 3DLabs, 3Dfx, TNT NVidia 2, și apar mereu altele noi.

Dispozitivele folosite pentru afișarea imaginii mediului virtual depind de tipul aplicației. În aplicații de realitate virtuală desktop sau de teleprezență, dispozitivul de afișare poate fi un singur display color, pe care utilizatorul urmărește imaginea mediului sintetic. În sistemele de realitate virtuală imersive se folosesc căști de vizualizare, astfel că utilizatorul vede numai imaginea sintetică, orice contact vizual cu realitatea fizică fiind complet întrerupt. În simulatoare de antrenament, imaginea se proiectează pe mai multe ecrane juxtapuse folosind mai multe proiectoare color.

### 1.3.2.2 Componente software de generare a imaginii vizuale

În generarea imaginii vizuale în realitatea virtuală, ca și în majoritatea aplicațiilor grafice, intervin mai multe componente software, care permit crearea sau redarea scenelor virtuale (fig. 1.4).

Programele de creare sau redare a scenelor virtuale (programe de aplicații) se dezvoltă pe baza unor sisteme de dezvoltare (toolkit-uri) sau direct, prin utilizarea unor biblioteci grafice care asigură interfața cu echipamentul hardware prin intermediul driverelor sistemului de operare.

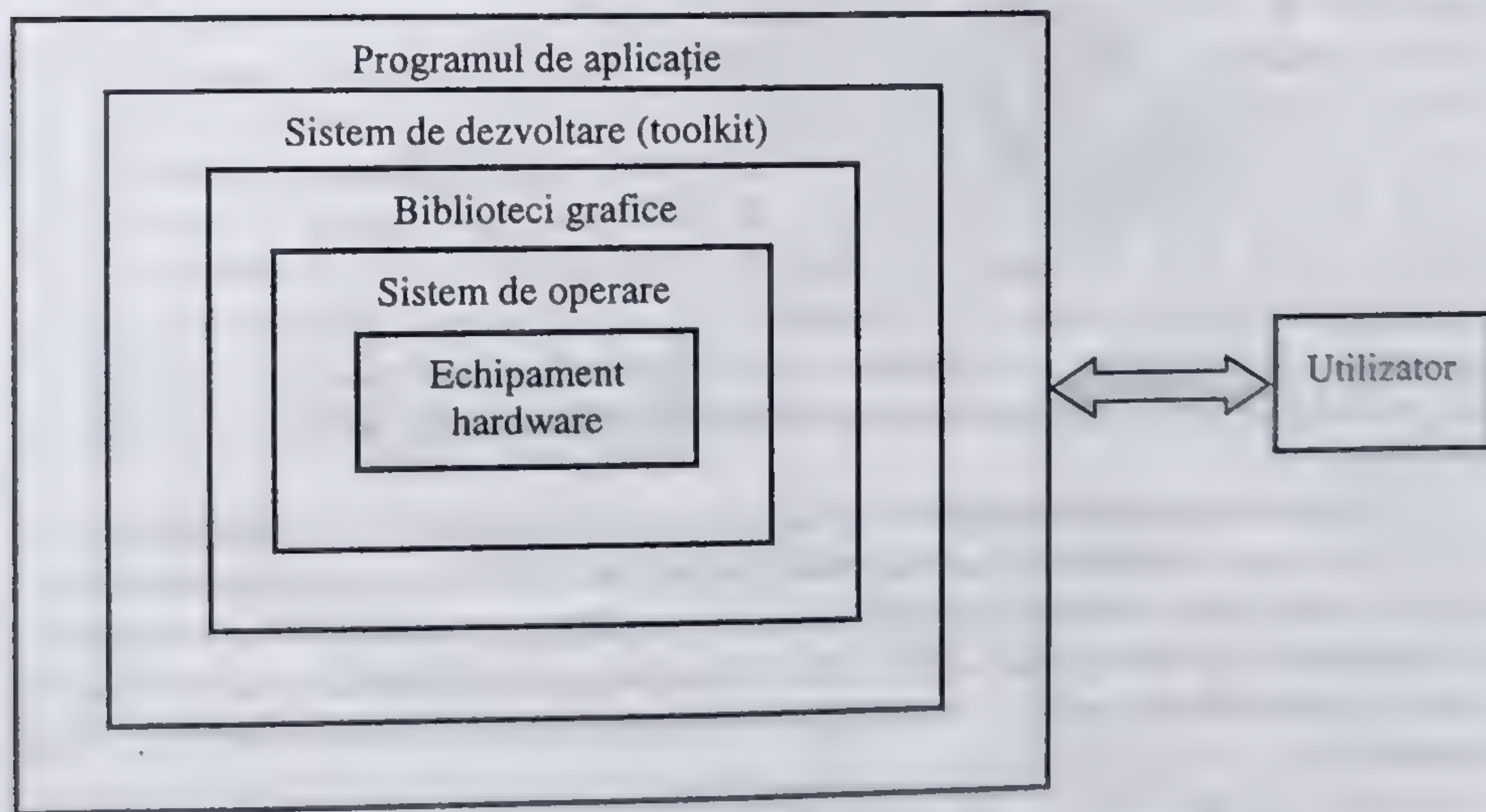


Fig. 1.4 Componentele software de generare a imaginii vizuale.

Sistemele de dezvoltare sunt de cele mai multe ori orientate către aplicație și prevăd un set de funcții de nivel înalt care permit crearea unui anumit tip de aplicație. De exemplu, există toolkit-uri pentru generarea obiectelor și a scenelor virtuale (3d Studio, Autocad, AC3d, Sense8, Designer Workbench, etc), toolkit-uri pentru redarea imaginii scenelor virtuale (Performer, EasyScene, browser CosmoPlayer, etc).



*Bibliotecile grafice* sunt pachete de funcții care asigură interfața programului de aplicație (creat direct sau prin intermediul unui toolkit care apelează funcțiile bibliotecii) cu echipamentele hardware ale sistemului grafic.

Bibliotecile grafice reprezintă nivelul de programare în care se încearcă introducerea portabilității programelor grafice, prin asigurarea unei interfețe independente de echipamentele hardware care să respecte anumite convenții de reprezentare a entităților grafice descrise în standarde.

Primul standard grafic a fost standardul GKS (*Graphical Kernel System*) elaborat de ISO în 1985, care conținea un set de funcții grafice 2D independente de echipament. Acest standard a fost extins în anul 1988 la GKS 3D, care conține funcții 3D independente de echipament.

Un alt standard, PHIGS (*Programmer's Hierarchical Graphical System*) este un standard 3D care permite în plus organizarea ierarhică a modelelor obiectelor și a scenelor virtuale.

Bibliotecile grafice cele mai generale sunt bibliotecile grafice care implementează un anumit standard în definirea funcțiilor de acces la echipamentele hardware. Cele mai cunoscute biblioteci grafice sunt OpenGL, Direct3D, QuickDraw, care sunt implementate în numeroase sisteme grafice.

În momentul de față, situația cea mai obișnuită în generarea imaginilor în aplicațiile de realitate virtuală este aceea în care se poate folosi un calculator dispunând de facilități grafice (adaptoare, acceleratoare sau subsisteme grafice, drivere, biblioteci), care permit programarea la nivel înalt, de cele mai multe ori complet independentă de dispozitivele hardware. Biblioteci grafice cum sunt OpenGL sau Direct3D asigură accesul la funcții grafice adaptate și optimizate pentru dispozitivele grafice disponibile ale sistemului. De aceea, în lucrare s-a acordat o mai mică întindere descrierii unor funcții existente în biblioteci (de exemplu funcțiile de generare a segmentelor de linie dreaptă, a cercurilor, etc.) și accentul s-a îndreptat către prezentarea metodelor de bază de generare a imaginilor și a modului în care acestea se pot aplica folosind limbaje și biblioteci grafice de nivel înalt.

Programele grafice propuse în această carte se prezintă sub forma de cod C, C++ și, uneori, pseudocod asemănător limbajului C. Biblioteca grafică OpenGL este folosită intens, datorită portabilității și disponibilității acesteia în aproape toate calculatoarele, de la calculatoare PC până la stații grafice puternice, sub un număr mare de sisteme de operare și apelabilă din numeroase sisteme de dezvoltare (toolkit).

### 1.3.2.3 Dispozitive de afișare

În sistemele de realitate virtuală se utilizează trei categorii de *dispozitive de afișare*: display-uri, proiectoare și dispozitive de afișare montate pe cap (*head mounted display* - HMD).

Display-urile și proiectoarele sunt folosite în sistemele de realitate virtuală neimersive sau semi-imersive, deoarece utilizatorul plasat în fața unei imagini afișate pe display sau proiectate pe un ecran poate să mai perceapă și elemente ale realității fizice. Imaginile afișate pe un singur display sau proiectate de un singur



proiector au un câmp de vizualizare redus, tipic 35-45°. Astfel de sisteme de afișare se folosesc în sistemele de realitate virtuală desktop, unde display-ul de afișare este chiar consola calculatorului.

În sistemele de realitate virtuală semi-imersive (cum sunt simulatoarele de zbor), un câmp de vizualizare atât de redus limitează posibilitatea de orientare a utilizatorului în mediul virtual și de aceea se folosesc mai multe display-uri juxtapuse sau proiectoare care proiectează imagini juxtapuse. Imaginea afișată de fiecare display sau proiector este un canal de imagine, iar imaginea totală a mediului virtual este o imagine multicanal.

Pentru generarea unor imagini multicanal se folosesc sisteme de vizualizare care permit crearea de imagini juxtapuse (acest aspect este explicat în capitolul 4) și dispozitive de afișare complexe, compuse din mai multe display-uri montate adecvat sau proiectoare și ecrane corespunzătoare.

În fig. 1.5 este prezentat un dispozitiv de afișare multicanal (cu 3 canale) compus din trei monitoare color montate juxtapus. Dispozitivele de afișare multicanal cu monitoare sau proiectoare sunt prevăzute cu posibilități de aliniere și reglare a imaginilor juxtapuse (*edge blending*).

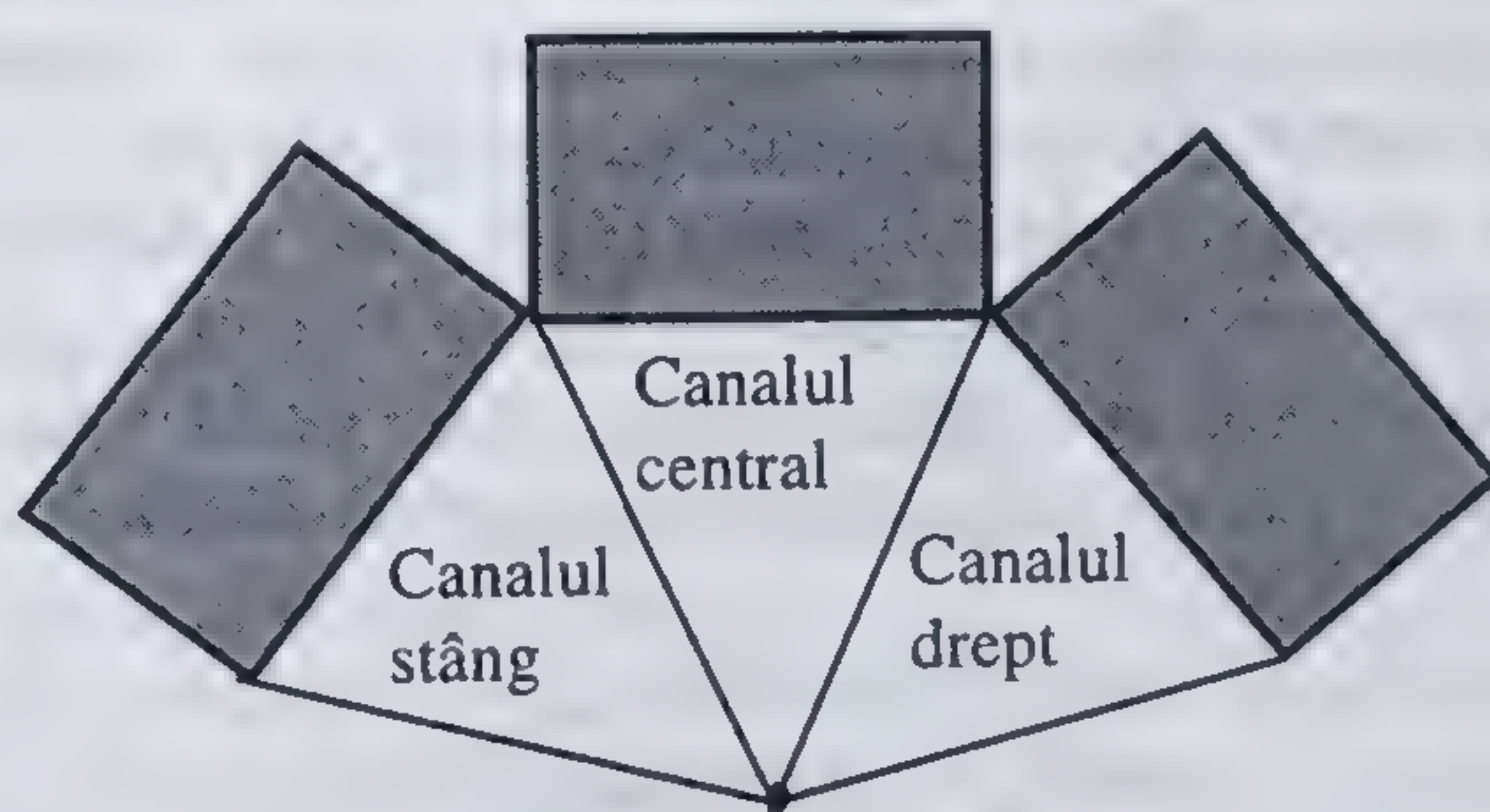


Fig. 1.5 Dispozitiv de afișare cu trei canale de imagine.

Dispozitivul de afișare montat pe cap (HMD), care se mai numește și cască de vizualizare, este unul dintre cele mai reprezentative dispozitive asociate realității virtuale, deoarece acesta permite imersiunea completă a utilizatorului în mediul virtual, prin percepția vizuală numai a imaginii sintetizate a mediului virtual. Acest dispozitiv folosește un fel de cască sau ochelari pentru a plasa în fața ochilor unul sau două display-uri video și un sistem optic cu lentile sau oglinzi care asigură focalizarea imaginii în câmpul vizual. În cele mai multe situații, imaginea percepută de un utilizator echipat cu o cască de vizualizare este imaginea generată pe display-ul căștii, dar există și căști de vizualizare cu sistem optic cu oglinzi semitransparente, care permit suprapunerea imaginii generate cu imaginea mediului real. Astfel de căști sunt folosite în realitatea îmbogățită.

Cele mai multe dintre dispozitivele HMD au două display-uri pentru afișarea imaginilor stereoscopice. Altele au un singur display mai mare pentru redarea cu rezoluție ridicată a imaginii video monoscopice.



Depărtarea obiectelor (adâncimea) poate fi percepută și în vederea monoculară, dar este mult mai bine percepută în vederea binoculară. În vederea monoculară, adâncimea este percepută prin intermediul acoperirii între obiecte, a umbrelor sau a paralaxei de mișcare, care înseamnă că obiectele mai apropiate par a se deplasa mai mult decât cele mai depărtate.

Cea mai completă percepție a adâncimii este obținută în vederea binoculară (stereoscopică), prin care pe retină se creează două imagini ale aceluiași obiect, ușor deplasate una față de alta. Creierul folosește diferența dintre poziția imaginilor create de cei doi ochi pentru a evalua depărtarea obiectului.

În sistemele de realitate virtuală, vederea stereoscopică este obținută prin plasarea a două display-uri, câte unul în fața fiecărui ochi. Imaginea generată pe fiecare display este calculată dintr-un punct de observare corespunzător ochiului respectiv. În același mod ca în vederea normală binoculară, creierul combină cele două imagini pentru a obține informația de depărtare a obiectelor. Un astfel de dispozitiv, numit dispozitiv de afișare stereo montat pe cap, mai necesită și un dispozitiv de captare a poziției care să furnizeze poziția capului utilizatorului în mediu.

Una din caracteristicile importante ale vederii stereoscopice este unghiul de vizibilitate binoculară (fig. 1.6) (*field of view*- FOV). Pentru unul din cele mai cunoscute dispozitive HMD stereoscopice, numit EyePhone și produs de firma VPL, unghiul de vizibilitate binoculară este de  $90^\circ$ , suficient de mare pentru a asigura senzația de imersie, iar unghiul de suprapunere este de  $60,6^\circ$ , suficient de mare pentru a asigura o percepție stereoscopică. Bineînțeles că dispozitive cu unghiuri de vizibilitate mai mari pot întări senzația de imersie.

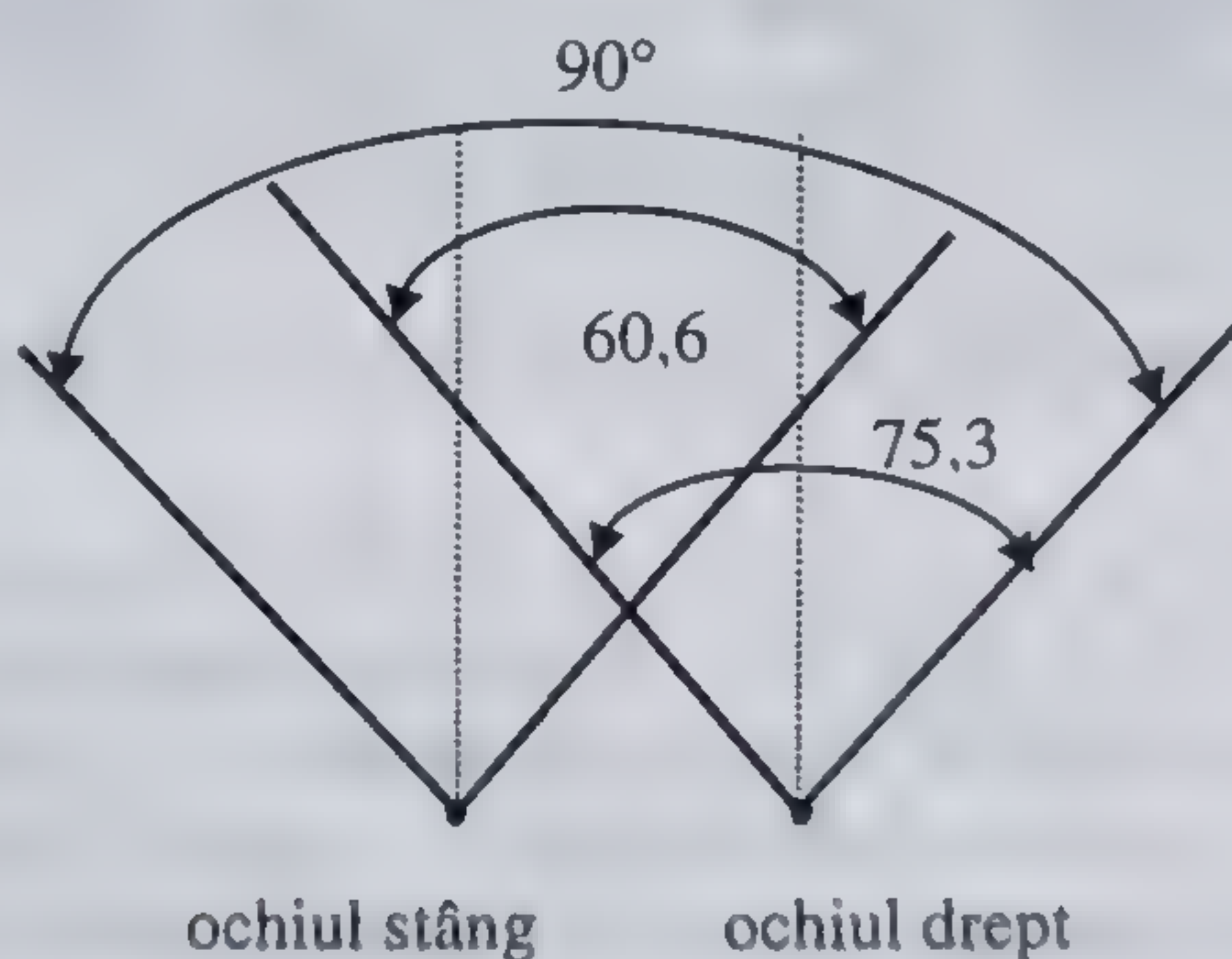


Fig. 1.6 Unghiul de vizibilitate binoculară ( $90^\circ$ ), de suprapunere ( $60,6^\circ$ ) și pentru un ochi ( $75,3^\circ$ ) a dispozitivului HMD EyePhone (VPL).

Dispozitivul *EyePhone* a fost unul din primele modele de cască de vizualizare prezente pe piață. Acesta era echipat cu display-uri Sony cu cristale lichide (*Liquid Crystal Display* - LCD) cu rezoluția de  $360 \times 240$  pixeli și avea un preț destul de ridicat (aproximativ 11 000 de dolari). În momentul de față, dispozitive HMD cu performanțe similare se pot găsi la prețuri mai scăzute, sub



1000 de dolari. Exemple de astfel de căști de vizualizare sunt Flight Helmet (produsă de Virtual Research Inc), sau Cyberface II (produsă de Leap Systems).

Dispozitivele HMD care folosesc display-uri LCD au o rezoluție scăzută, ceea ce face ca imaginea afișată să fie de slabă calitate. Soluția pentru obținerea unei rezoluții mai mari este de a se folosi display-uri cu tub catodic (*Cathode Ray Tube-CRT*) în locul celor cu cristale lichide. Astfel de dispozitive echipate cu display-uri CRT sunt disponibile de mai multă vreme pe piață, de exemplu HMD-131, produs de compania Virtual Reality Inc., monocrom, cu rezoluție de  $1280 \times 1024$  pixeli, sau Datavision 9c, produs de firma n-Vision, cu display-uri color și rezoluție de  $1280 \times 960$  pixeli. Prețul unor astfel de dispozitive este însă destul de ridicat (în jur de 70 000 dolari).

### 1.3.3 GENERAREA SUNETULUI

Sistemele de realitate virtuală cu un grad puternic de imersivitate posedă dispozitive de generare a sunetului virtual tridimensional. Sunetul virtual nu trebuie să fie confundat cu sunetul stereo. Sunetul stereo, generat de două căști de sunet fixate pe urechi, nu se modifică atunci când utilizatorul mișcă (rotește) capul, ceea ce dă senzația că sursa de sunet se mișcă odată cu utilizatorul. Sunetul virtual este astfel generat încât se modifică atunci când utilizatorul mișcă capul, ceea ce creează senzația că sursa sonoră își păstrează poziția în spațiu.

Căștile prin care se generează sunetul virtual sunt *căști binaurale* prevăzute cu dispozitiv de captare a poziției. Pentru calculul semnalului sonor virtual se folosesc datele de poziție a capului, furnizate de dispozitivul de captare, și datele de poziție și caracteristicile sursei sonore.

Generatoarele de sunet virtual sunt implementate în sistemele de realitate virtuală prin module echipate cu procesoare de semnal. Mai multe tipuri de generatoare de sunet virtual au fost realizate de compania Crystal River Engineering, la prețuri care variază de la 1000 la 60 000 de dolari. Modelele mai puternice de generatoare de sunet (de exemplu, modelul Acoustetron) permit redarea sunetelor unui număr mare de surse sonore (mai mult de 16 surse sonore).

### 1.3.4 SIMULAREA SENZAȚIEI TACTILE ȘI A FORȚEI DE REACȚIE

Multe din sistemele de realitate virtuală comerciale se bazează pe imaginea vizuală a mediului înconjurător, neglijând aspectul fizic al realității: identificarea tactilă a obiectelor și reacția acestora la atingere. Există însă aplicații ale realității virtuale în care simularea contactului cu obiectele mediului virtual este importantă și nu mai poate fi neglijată, un exemplu tipic fiind simulatoarele chirurgicale. Este nedorit și riscant ca un medic să fie antrenat pentru execuția unei operații chirurgicale, fără să aibă senzația atingerii organului investigat. În astfel de aplicații este necesară creșterea gradului de realism al mediului sintetic prin simularea caracteristicilor fizice ale obiectelor: greutatea, rigiditatea, compoziția suprafeței, ș.a.m.d.



Din punct de vedere al terminologiei, în [Burd97] se precizează doi termeni care descriu două senzații diferite care intervin la atingerea obiectelor: senzația tactilă (*touch feedback*) și forța de reacție (*force feedback*). Senzația tactilă este senzația recepționată de piele atunci când este atinsă mecanic, termic, chimic sau electric. Senzorii tactili sunt plasați la suprafața pielii, cea mai mare densitate a acestora găsindu-se în mână și furnizează informații despre rugozitatea suprafețelor sau temperatură. În realitatea virtuală se simulează senzația tactilă prin acționarea cu o forță repartizată spațial asupra extremității degetelor.

Forța de reacție este răspunsul unui obiect la o acțiune externă, de exemplu rezistența la apăsare sau greutatea pe care o prezintă dacă este ridicat. Forța de reacție este recepționată de senzori plasați în interiorul corpului, în general în tendoanele musculare. În realitatea virtuală, forța de reacție se generează prin dispozitive care produc o forță de apăsare asupra organismului. În cazul obiectelor virtuale atinse cu mâna, se generează o forță de apăsare asupra degetelor corespunzătoare reacției obiectelor atinse.

În afară de termenii descriși, se mai folosesc termenii de reacție haptică (*haptic feedback*, de la termenul grecesc *haptesthai*, sinonim cu simțul tactil) sau reacție chinestezică (*kinesthetic feedback*).

Interacțiunea utilizatorului cu obiectele mediului virtual este asigurată prin intermediul unui dispozitiv numit mănușă de date (*data glove*). O mănușă de date (sau mănușă senzitivă) este prevăzută cu senzori ai mișcării și poziției mâinii și cu dispozitive pneumatice pentru generarea forței de reacție la atingerea obiectelor.

Diferite modele de mănuși de date sunt deja disponibile, de exemplu, Rutgers Portable Master, realizată la Universitatea Rutgers [Burd92], LRP Portable Master, realizată la Laboratorul de Robotică din Paris (Laboratoire de Robotique de Paris - LRP), sau Force ArmMaster, realizată de compania EXOS Inc.

## 1.4 SISTEME DE REFERINȚĂ TRIDIMENSIONALE

Pentru crearea și redarea scenelor virtuale este necesar ca obiectele să fie poziționate într-un sistem de referință tridimensional. Există mai multe posibilități de a specifica poziția unei mulțimi de puncte (vârfuri) prin care este reprezentat un obiect în spațiul tridimensional: coordonate cilindrice, coordonate sferice, coordonate carteziane. Dintre aceste sisteme de referință, cel mai utilizat în aplicațiile grafice este sistemul de coordonate cartezian.

Sistemul de coordonate cartezian în care sunt definite toate obiectele scenei virtuale se numește sistem de referință universal (*world coordinate system*- WCS).

Un sistem de coordonate cartezian se definește prin originea O și trei axe perpendiculare, Ox, Oy și Oz, orientate după regula mâinii drepte sau după regula mâinii stângi. Diferența dintre cele două sisteme se poate urmări în fig. 1.7.

Într-un sistem orientat după regula mâinii drepte, dacă se rotește mâna dreaptă în jurul axei z de la axa x pozitivă spre axa y pozitivă, orientarea degetului



mare este în direcția  $z$  pozitiv. Într-un sistem orientat după regula mâinii stângi, rotirea de la axa  $x$  pozitivă spre axa  $y$  pozitivă, cu orientarea degetului mare în direcția  $z$  pozitiv, se obține folosind mâna stângă. Orientarea după regula mâinii drepte a sistemelor de coordonate corespunde convenției matematice standard.

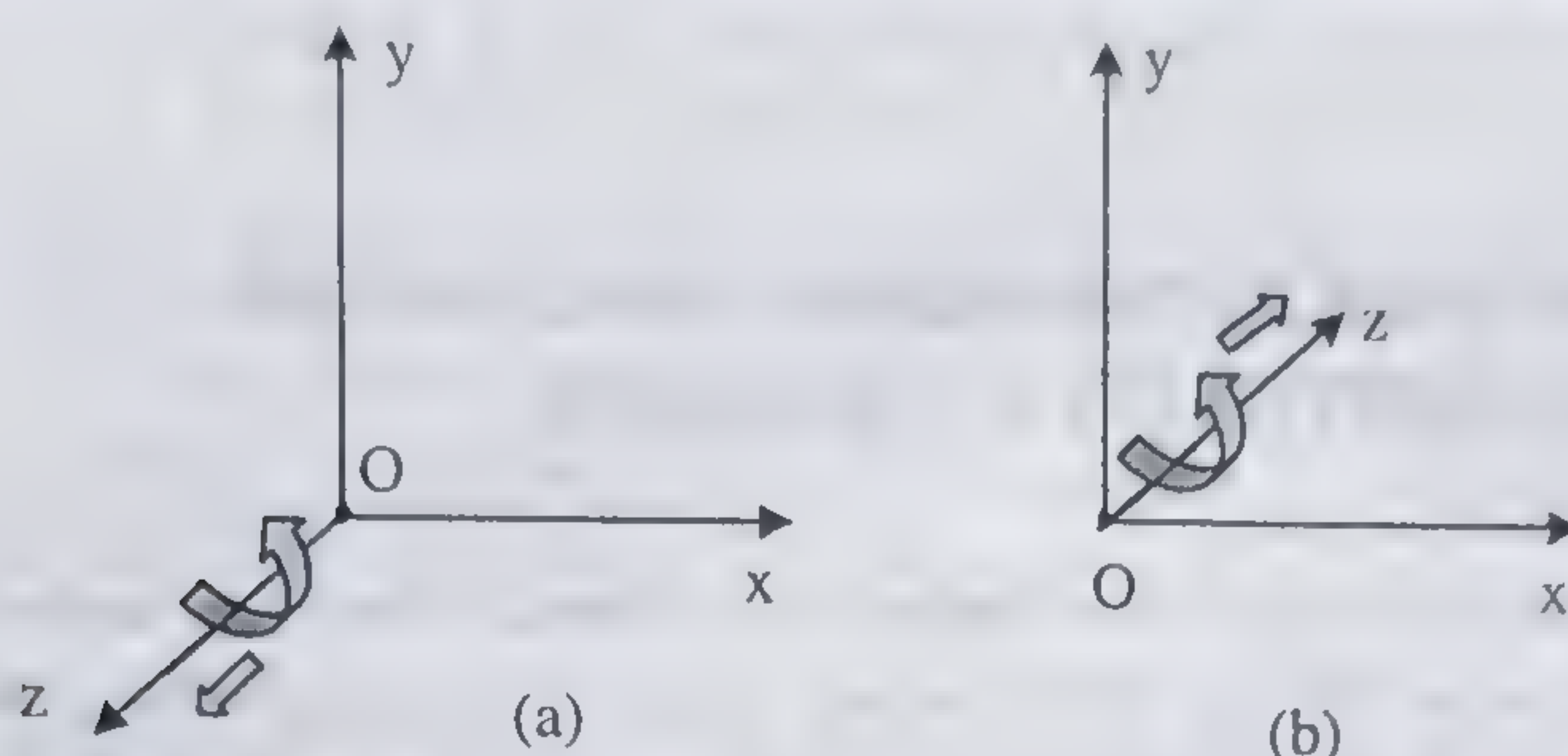


Fig.1.7 (a) Sistem de coordonate orientat după regula mâinii drepte și (b) după regula mâinii stângi.

Într-un sistem orientat după regula mâinii drepte, dacă se rotește mâna dreaptă în jurul axei  $z$  de la axa  $x$  pozitivă spre axa  $y$  pozitivă, orientarea degetului mare este în direcția  $z$  pozitiv. Într-un sistem orientat după regula mâinii stângi, rotirea de la axa  $x$  pozitivă spre axa  $y$  pozitivă, cu orientarea degetului mare în direcția  $z$  pozitiv, se obține folosind mâna stângă. Orientarea după regula mâinii drepte a sistemelor de coordonate corespunde convenției matematice standard.

Diferite sisteme de grafică tridimensională sau de realitate virtuală folosesc convenții diferite pentru definirea sistemelor de referință, ceea ce conduce la confuzii, dacă nu se precizează convenția folosită. În acest text, pentru sistemul de referință universal se folosește convenția de sistem de coordonate drept. În grafica tridimensională se mai folosesc și alte sisteme de referință, care permit descrierea operațiilor de transformări geometrice și care vor fi precizate pe parcursul lucrării.

Un punct  $P$  în spațiul tridimensional se reprezintă în sistemul de referință cartezian printr-un triplet de valori scalare  $x, y, z$ , care reprezintă componentele vectorului de poziție  $\mathbf{OP}$  pe cele trei axe de coordonate. Dacă se notează cu  $\mathbf{i}, \mathbf{j}, \mathbf{k}$  versorii (vectorii unitate) ai celor trei axe de coordonate  $x, y, z$ , atunci vectorul de poziție al punctului  $P$  este  $\mathbf{OP} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ . În notația matriceală, un punct în spațiul tridimensional se poate reprezenta printr-o matrice linie sau coloană:

$$\mathbf{P} = [x \quad y \quad z] \text{ sau } \mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Ambele convenții sunt folosite în egală măsură în sistemele grafice, ceea ce, din nou, poate provoca diferite confuzii, dacă nu se precizează convenția folosită. Convenția de reprezentare sub formă de matrice linie a unui punct, folosită în unele lucrări [Watt95], [Mold96], are avantajul că exprimă operațiile de



conecatenare a matricelor într-un mod natural, de la stânga la dreapta. Convenția de reprezentare matematică, standardul grafic PHIGS, biblioteca grafică OpenGL, ca și unele din lucrările de referință în domeniu [Foley90], folosesc notația de matrice coloană pentru un punct în spațiul tridimensional, care este adoptată și în lucrarea prezentă.

## 1.5 REPREZENTAREA CULORILOR ÎN SISTEMELE GRAFICE

Imaginea care se obține pe display este compusă dintr-un anumit număr de pixeli, dat de rezoluția display-ului. Fiecare pixel are o poziție pe ecran, dată de adresa lui în fereastra de afișare, și o culoare care poate fi reprezentată în mai multe modele: modelul RGB, modelul HSV, modelul HLS, și altele. Dintre aceste modele, în grafică se folosește cel mai frecvent modelul RGB.

### 1.5.1 MODELUL RGB

În modelul RGB, culoarea este reprezentată printr-un triplet de culori primare, roșu (*red*) verde (*green*), albastru (*blue*). Utilizarea preponderentă a modelului RGB în grafică se datorează în primul rând faptului că monitoarele color folosesc acest model de producere a culorii. Culoarea este produsă pe ecranul monitorului prin excitarea a trei puncte adiacente de substanță fosforescentă de culori roșu, verde și, respectiv, albastru. Punctele fosforescente fiind apropiate, ochiul percepe tripletul de puncte ca un singur punct a cărui culoare este suma celor trei componente. Spațiul de reprezentare a tuturor culorilor în modelul RGB este un cub într-un sistem de coordonate cu axele notate Roșu (*Red*), Verde (*Green*), Albastru (*Blue*) (fig. 1.8).

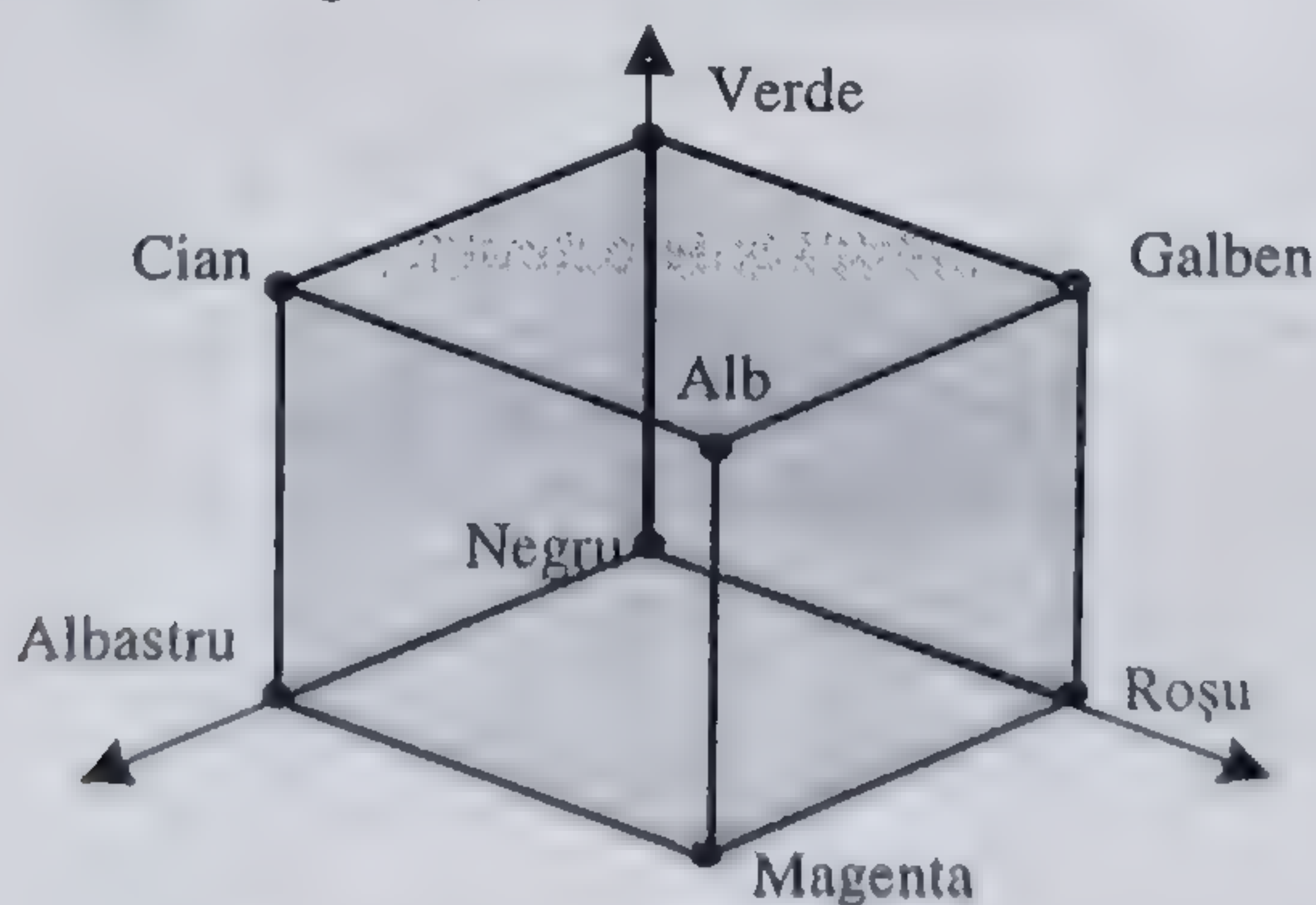


Fig. 1.8 Spațiul culorilor în modelul RGB.

Orice culoare în modelul RGB se exprimă printr-un triplet  $(r, g, b)$  și îi corespunde un punct în spațiul RGB al cărui vector  $C$  este:



$$C = rR + gG + bB \quad (1.1)$$

unde  $R$ ,  $G$ ,  $B$  sunt versorii axelor roșu (*red*), verde (*green*), albastru (*blue*). În acest model culoarea negru este reprezentată prin tripletul  $(0,0,0)$ , iar culoarea alb este reprezentată prin tripletul  $(1,1,1)$ .

În sistemele grafice se mai folosește o variantă a modelului RGB, modelul RGBA, unde cea de-a patra componentă ( $\alpha$ ) indică transparența suprafeței. Valoarea 1 a acestei componente ( $\alpha = 1$ ) înseamnă suprafață opacă, iar valoarea minimă ( $\alpha = 0$ ) înseamnă suprafață complet transparentă. Dacă transparența unei suprafețe este diferită de zero, atunci culoarea care se atribuie pixelilor acestei suprafețe se modifică în funcție de culoarea existentă în bufferul de imagine în pozițiile corespunzătoare. Culoarea rezultată a unui pixel este  $C' = f(\alpha, C, C_p)$ , unde:

- $C(r, g, b)$  este culoarea suprafeței
- $\alpha$  este transparența suprafeței, cu valori între 0 și 1
- $C_p(r_p, g_p, b_p)$  este culoarea precedentă a pixelului, aflată în bufferul de imagine
- $C'(r', g', b')$  este noua culoare a pixelului, rezultată prin aplicarea unei funcții  $f$  fiecărei componente de culoare.

Bibliotecile grafice permit definirea mai multor funcții  $f$  de combinare liniară între culoarea transparentă cu culoarea de fond.

### 1.5.2 MODELUL HSV

Modelul HSV reprezintă o transformare neliniară a spațiului RGB, prin care culorile sunt specificate prin componentele: nuanță (*hue*), saturație (*saturation*) și valoare (*value*). Modelul HSV este folosit în definirea mai intuitivă a culorilor (perceptuală), decât definirea prin culori primare. De exemplu, sarcina de a modifica o culoare astfel încât să fie “mai luminosă”, “mai galbenă”, etc, se realizează mult mai simplu în modelul HSV decât în modelul RGB.

Spațiul culorilor în modelul HSV este o piramidă hexagonală într-un sistem de coordonate polare HVS (fig. 1.9). Definițiile parametrilor H, S, V sunt legate de modul în care este percepută culoarea:

- Nuanța este acea calitate prin care ochiul distinge o familie de culori de altă familie de culori, de exemplu roșu de galben, verde de albastru.
- Saturația, numită și cromaticitate (*chroma*), este calitatea culorii prin care se distinge o culoare puternică de o culoare slabă. Se mai poate defini ca intensitatea culorii sau distanța față de nuanța gri.
- Valoare este calitatea prin care se distinge o culoare luminoasă de o culoare întunecată.

În modelul HVS, variația parametrului H (între 0 și 360°) corespunde selecției culorii. Scăderea parametrului S (desaturarea culorii) înseamnă adăugare de alb. Scăderea parametrului V (devaluarea culorii) înseamnă adăugare de negru.



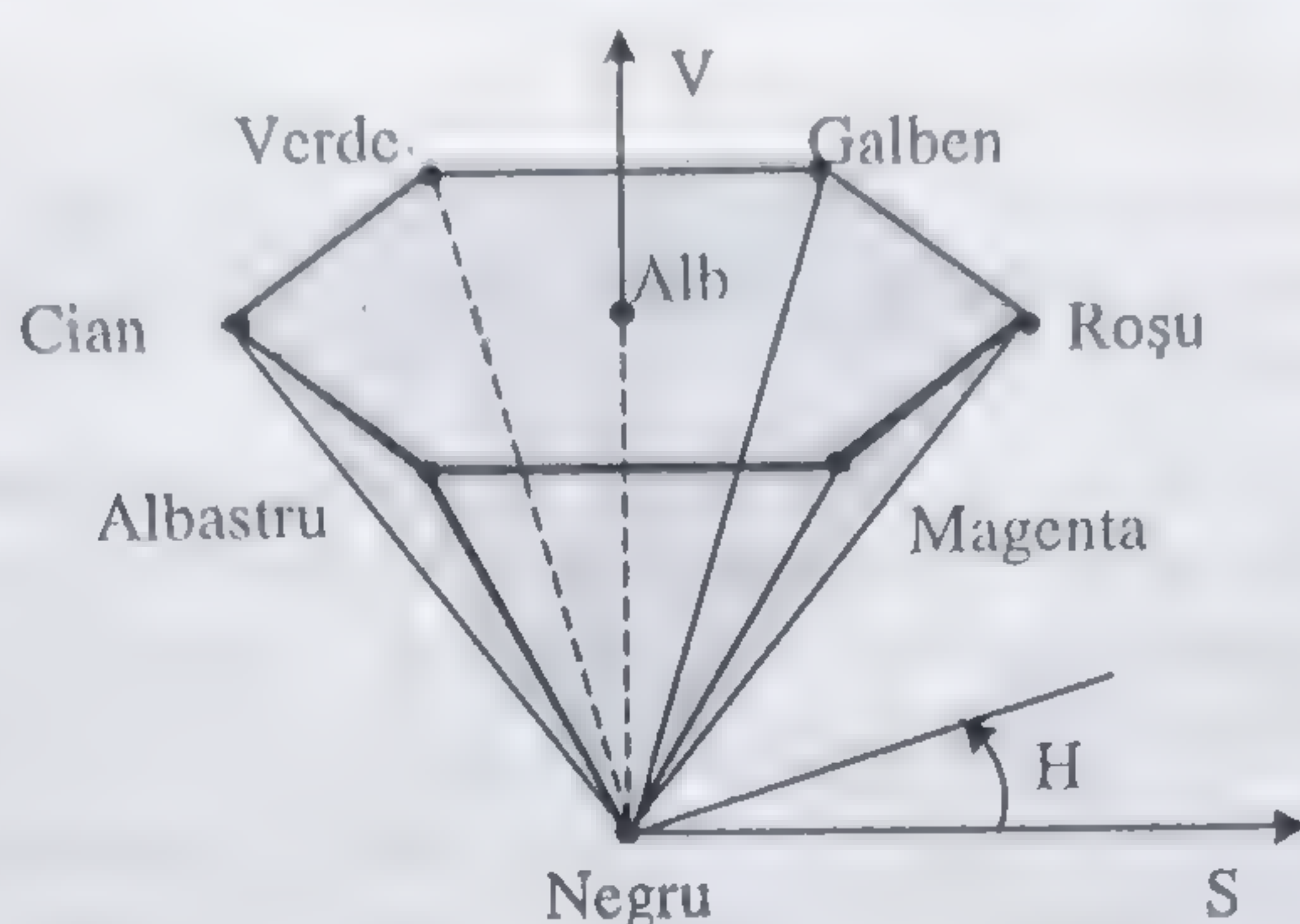


Fig.1.9 Spațiul culorilor în modelul HSV.

Conversia din spațiul RGB în spațiul HSV se înțelege mai ușor într-o interpretare geometrică: hexagonul din reprezentarea HVS se obține prin proiectarea cubului RGB pe un plan normal la diagonala principală, pe direcția diagonalei principale. Corespondența dintre vârfurile hexagonului de bază HSV și vârfurile cubului RGB este:

RGB		HSV
100	roșu	0,1,1 (H este măsurat în grade)
110	galben	60,1,1
010	verde	120,1,1
011	cian	180,1,1
001	albastru	240,1,1
101	magenta	300,1,1

Pentru aceeași definiție a unei culori (de exemplu, prin tripletul rgb) este posibil ca monitoare diferite să genereze culori diferite, datorită caracteristicilor constructive diferite. În general, între componenta de culoare transmisă unui monitor și intensitatea produsă de ecranul acestuia există relația neliniară:

$$R_m = K(R_i)^{\chi_r}; G_m = K(G_i)^{\chi_g}; B_m = K(B_i)^{\chi_b} \quad (1.2)$$

unde  $\chi$  ia valori între 2.3 și 2.8.

Liniașizarea relației între valorile componentelor culorii produse prin program și intensitățile generate pe ecran se numește corecție gamma (*gamma corection*). Corecția gamma se realizează aplicând o transformare care, compusă cu transformarea (1.2), să asigure o relație liniară între culoarea ( $R_i, G_i, B_i$ ) generată prin program și culoarea ( $R_m, G_m, B_m$ ) generată pe ecranul monitorului.

Acestă transformare este:

$$R_i' = k(R_i)^{1/\chi_r}; G_i' = k(G_i)^{1/\chi_g}; B_i' = k(B_i)^{1/\chi_b} \quad (1.3)$$

Corecția gamma se poate implementa printr-o tabelă de căutare inclusă în controlerul grafic, care transformă valorile  $R_i, G_i, B_i$  citite din bufferul de imagine în valorile corespunzătoare  $R_i', G_i', B_i'$  transmise monitorului. Lipsa corecției gamma, sau o corecție gamma calculată, eronat produce imagini incorecte.



## MODELAREA OBIECTELOR

Modul cel mai convenabil de creare a scenelor virtuale este acela în care fiecare obiect este modelat într-un sistem de coordonate propriu, numit sistem de referință model (sau sistem de referință local), în care punctele (vârfurile) obiectului sunt precizate relativ la un anumit punct de referință local. De fapt, în modelarea ierarhică, un obiect complex poate avea un număr oarecare de sisteme de referință locale, câte unul pentru fiecare parte componentă a sa. Instanțierea unui obiect în scena virtuală înseamnă amplasarea acestuia în sistemul de referință universal printr-o succesiune de scalări, rotații și translații, care transformă obiectul din sistemul de referință local în sistemul de referință universal. Această succesiune de transformări este cunoscută sub numele de transformare de modelare.

Proprietățile obiectelor tridimensionale care se modelează în aplicațiile grafice se pot împărți în două categorii: *forma* și *atribute de aspect*. Informația de formă a unui obiect este diferită de celelalte atribute ale obiectului, deoarece forma este aceea care determină modul în care obiectul apare în redarea grafică și toate celelalte atribute se corelează cu forma obiectului (de exemplu, culoarea se specifică pentru fiecare element de suprafață a obiectului).

Din punct de vedere al formei, obiectele tridimensionale reprezentate în grafica pe calculator pot fi obiecte *solide* sau obiecte *deformabile*. Un solid este un obiect tridimensional a cărui formă și dimensiuni nu se modifică în funcție de timp sau de poziția în scenă (proprietatea de formă volumetrică invariantă). Majoritatea aplicațiilor de realitate virtuală se bazează pe scene compuse din solide, dar există și aplicații în care obiectele reprezentate își modifică forma și dimensiunile într-un mod predefinit sau ca urmare a unor acțiuni interactive (de exemplu, în simulări ale intervențiilor chirurgicale). Chiar și reprezentarea unor astfel de obiecte (obiecte deformabile) se bazează pe un model al unui solid care se modifică în cursul experimentului de realitate virtuală. În lucrarea de față se vor prezenta modele ale solidelor care stau la baza prelucrărilor din grafică și realitate virtuală.

Modelarea solidelor este o tehnică de proiectare, vizualizare și analiză a modului în care obiectele reale se reprezintă în calculator. În ordinea importanței și a frecvenței de utilizare, metodele de modelare și reprezentare a obiectelor sunt următoarele:



1. *Modelarea poligonală.* În această formă de reprezentare, obiectele sunt approximate printr-o rețea de fețe care sunt poligoane planare.
2. *Modelarea prin rețele de petice parametrice bicubice (bicubic parametric patches).* Obiectele sunt approximate prin rețele de elemente spațiale numite petice (patches). Acestea sunt reprezentate prin polinoame cu două variabile parametrice, în mod obișnuit cubice.
3. *Modelarea prin compunerea obiectelor (Constructive Solid Geometry - CSG).* Obiectele sunt reprezentate prin colecții de obiecte elementare, cum sunt cilindri, sfere, poliedre.
4. *Modelarea prin divizare spațială.* Obiectele sunt încorporate în spațiu, prin atribuirea unei etichete fiecărui element spațial, în funcție de obiectul care ocupă elementul respectiv.

Aceste metode de modelare și reprezentare a solidelor se pot grupa în reprezentări prin suprafață de frontieră (primele două metode) și reprezentări prin volum (ultimele două metode).

## 2.1 MODELAREA POLIGONALĂ A OBIECTELOR

Modelarea poligonală, în care un obiect constă dintr-o rețea de poligoane planare care aproximează suprafața de frontieră (*boundary representation – B-rep*), este forma “clasică” folosită în grafica pe calculator. Motivele utilizării extinse a acestei forme de reprezentare sunt ușurința de modelare și posibilitatea de redare rapidă a imaginii obiectelor. Pentru obiectele reprezentate poligonal s-au dezvoltat algoritmi de redare eficienți, care asigură calculul umbririi, eliminarea suprafețelor ascunse, texturare, anti-aliasing, frecvent implementați hardware în sistemele grafice. În reprezentarea poligonală, un obiect tridimensional este compus dintr-o colecție de fețe, fiecare față fiind o suprafață plană reprezentată printr-un poligon.

### 2.1.1 REPREZENTAREA POLIGOANELOR

Un poligon este o regiune din plan mărginită de o colecție finită de segmente de dreaptă care formează un circuit închis simplu.

Fie  $n$  puncte în plan, notate  $v_0, v_1, \dots, v_{n-1}$  și  $n$  segmente de dreaptă  $e_0 = v_0v_1, e_1 = v_1v_2, \dots, e_{n-1} = v_{n-1}v_0$ , care conectează perechi de puncte succesive în ordine ciclică, deci inclusiv conexiunea între ultimul punct și primul punct din listă. Aceste segmente mărginesc un poligon, dacă și numai dacă:

- (a) Intersecția fiecărei perechi de segmente adiacente în ordinea ciclică este un singur punct, conținut de ambele segmente:  $e_i \cap e_{i+1} = v_{i+1}$ , pentru oricare  $i = 0, \dots, n-1$ .
- (b) Segmente neadiacente nu se intersectează:  $e_i \cap e_j = \emptyset$ , pentru orice  $j \neq i+1$ .



Segmentele care mărginesc un poligon (linia poligonală) formează un circuit închis (*ciclu*), deoarece segmentele sunt conectate capăt la capăt și ultimul segment conectează ultimul punct cu primul punct; ciclul este simplu deoarece segmentele neadiacente nu se intersectează.

Punctele  $v_i$  se numesc vârfurile poligonului (*vertices*); segmentele  $e_i$  se numesc muchii (sau laturi) ale poligonului. De remarcat că un poligon conține  $n$  vârfuri și  $n$  muchii și că muchiile sunt orientate, astfel încât formează un ciclu (circuit închis). O astfel de orientare a segmentelor se numește orientare consistentă. În general, se folosește ordinea de parcurgere în sensul invers acelor de ceasornic: dacă se parcurg muchiile în sensul lor de definiție, interiorul poligonului este văzut întotdeauna în partea stângă (fig. 2.1).

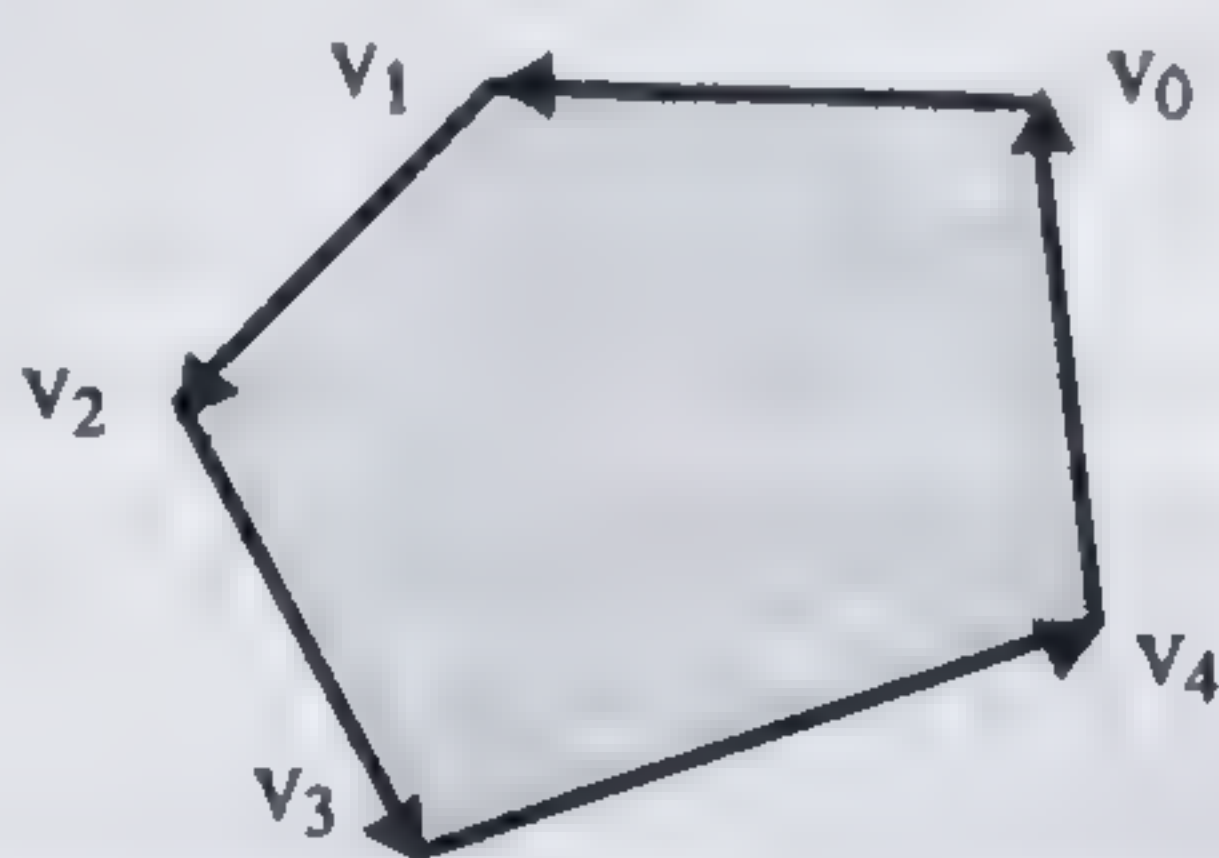


Fig. 2.1 Segmentele liniei poligonale sunt orientate și nu se autointersectează.

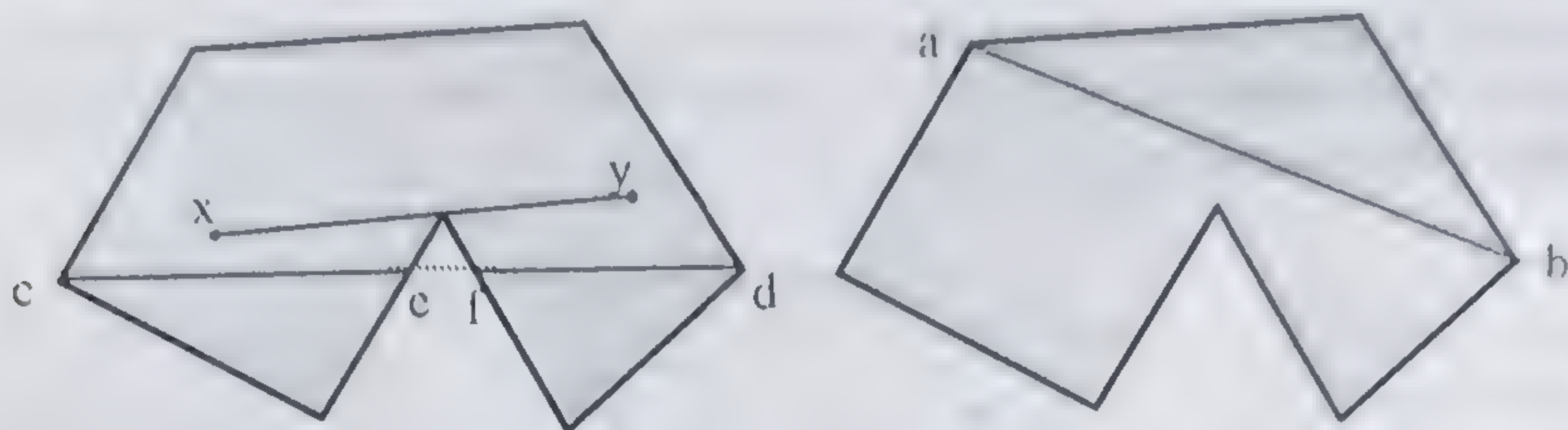
O teoremă importantă în prelucrarea poligoanelor este teorema lui *Jordan*, care spune că orice curbă plană închisă simplă împarte planul în două părți: o parte interioară curbei, care este o regiune limitată, și o parte exterioară curbei, care este o regiune nelimitată. Deși pare simplă din punct de vedere intuitiv, demonstrația teoremei lui Jordan este destul de dificilă și poate fi găsită în referințele bibliografice [Rour93].

Această teoremă justifică definiția care se mai folosește pentru poligoane, și anume: poligonul este o regiune limitată din plan, mărginită de o colecție de segmente orientate consistent. Prin această definiție se consideră poligonul ca o regiune închisă din plan. Uneori, poligonul este considerat ca fiind format numai din conturul său, deci numai de segmentele de dreaptă care mărginesc regiunea, și nu de regiunea însăși. În continuare, se folosește notația  $\partial P$  pentru a desemna conturul poligonului  $P$  (linia poligonală închisă care mărginește poligonul).

**Triangularizarea poligoanelor.** O altă proprietate importantă a poligoanelor este proprietatea de triangularizare. Se demonstrează că orice poligon poate fi împărțit în triunghiuri prin adăugarea a zero sau mai multe diagonale. Proprietatea de triangularizare se bazează pe noțiunile de vizibilitate și diagonală în poligoane. Un punct  $x$  din interiorul unui poligon este vizibil pentru un alt punct  $y$ , dacă și numai dacă segmentul  $xy$  nu este în nici un punct al său exterior poligonului, adică  $xy \subseteq P$ . Acest lucru înseamnă că linia care unește două puncte  $x$  și  $y$ , vizibile unul altuia, poate atinge un vârf al poligonului. Vizibilitatea între două puncte  $x$  și  $y$  este completă (*clearly visible*), dacă linia care unește cele două puncte nu atinge frontiera poligonală (fig. 2.2).



O diagonală a unui poligon este un segment de dreaptă între două vârfuri  $a$  și  $b$ , complet vizibile unul altuia. Acest lucru înseamnă că intersecția dintre segmentul închis  $ab$  și  $\partial P$  este mulțimea  $\{a, b\}$ , adică segmentul  $ab$  nu atinge linia poligonală  $\partial P$  în alte puncte decât vârfurile  $a$  și  $b$ , de început și de sfârșit ale segmentului. Condițiile ca segmentul  $ab$  care unește vârfurile  $a$  și  $b$  ale unui poligon să fie o diagonală în acel poligon sunt deci:  $ab \subseteq P$  și  $ab \cap \partial P = \{a, b\}$ . Orice diagonală împarte un poligon în două poligoane mai mici (fig. 2.2).



**Fig. 2.2** Vizibilitate și diagonale în poligoane.

Punctele  $x$  și  $y$  sunt vizibile unul altuia.

Punctele  $c$  și  $d$  sunt invizibile unul altuia deoarece  $ef \not\subseteq P$ .

Punctele  $a$  și  $b$  sunt complet vizibile unul altuia, deci  $ab$  este diagonală în  $P$ .

Două diagonale ale unui poligon sunt neîncrucișate (*noncrossing*) dacă intersecția lor este o submulțime a capetelor lor (punctele de început și de sfârșit ale segmentelor). Dacă se adaugă atâtea diagonale neîncrucișate câte sunt posibile într-un poligon, atunci poligonul este împărțit în triunghiuri. O astfel de partiționare a unui poligon în triunghiuri se numește *triangularizarea* poligonului. Diagonalele se pot adăuga în orice ordine, atât timp cât sunt neîncrucișate. Demonstrația teoremei conform căreia orice poligon admite o triangularizare se bazează pe teorema lui Meister, demonstrată în [Rour93], care stabilește că orice poligon cu  $n \geq 4$  vârfuri admite cel puțin o diagonală.

Teorema triangularizării se bazează și pe lema numărului de diagonale: Orice triangularizare a unui poligon  $P$  cu  $n$  vârfuri utilizează  $n - 3$  diagonale și constă din  $n - 2$  triunghiuri. Aceste teoreme se demonstrează prin inducție. În fig. 2.3 este prezentată triangularizarea unui poligon convex cu opt laturi; se inserează  $8 - 3 = 5$  diagonale neîncrucișate și rezultă  $8 - 2 = 6$  triunghiuri.

Teorema triangularizării, care asigură că orice poligon poate fi divizat în triunghiuri, reprezintă suportul celei mai eficiente metode de generare (redare) a imaginii obiectelor tridimensionale: obiectele se reprezintă prin fețe poligonale, fiecare poligon se descompune în triunghiuri și triunghiurile sunt generate prin algoritmi implementați hardware.

Din punct de vedere al reprezentării în program a poligoanelor, cea mai compactă formă este reprezentarea printr-o listă liniară de vârfuri, fiecare vârf fiind specificat printr-o structură (sau clasă, în programarea orientată pe obiecte) care memorează (cel puțin) coordonatele vârfului. Alte date referitoare la vârfurile poligoanelor necesare în modelarea și redarea obiectelor (normală, culoare, coordonate de texturare, etc.) vor fi descrise în capitolele care urmează.



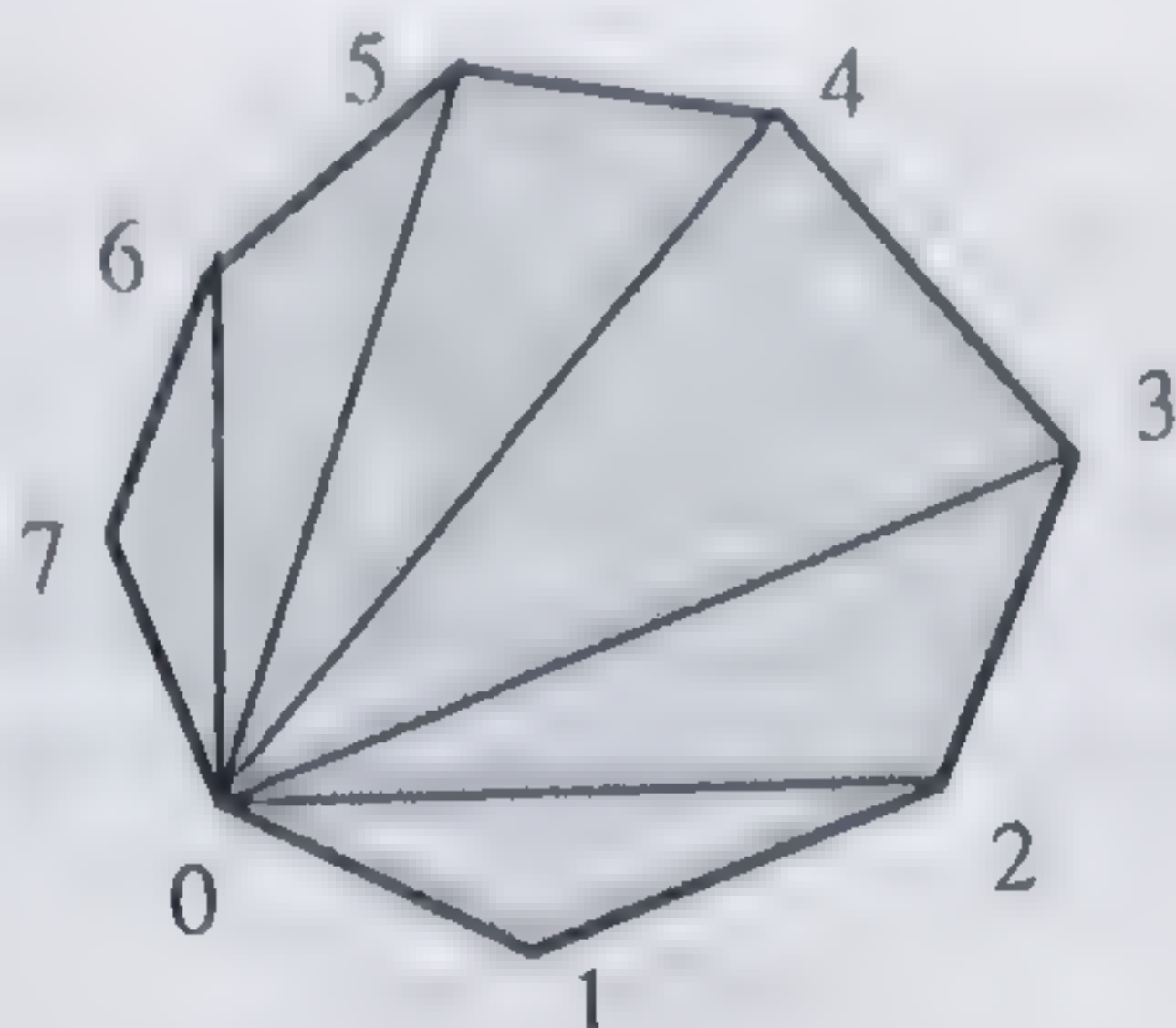


Fig. 2.3 Triangularizarea unui poligon convex.

Este posibilă reprezentarea unui poligon și prin lista segmentelor sale, dar această reprezentare necesită un volum mai mare de date și este folosită în implementarea anumitor algoritmi de prelucrare a poligoanelor (reuniune, divizare, etc.) și mai puțin în reprezentarea modelului unui obiect. Lista liniară de vârfuri poate fi implementată ca vector sau ca listă simplu sau dublu înlănțuită.

### 2.1.2 REPREZENTAREA POLIEDRELOR

În modelarea și reprezentarea prin suprafața de frontieră, obiectele sunt approximate prin poliedre și modelul lor este reprezentat prin suprafața poliedrului, compusă dintr-o colecție de poligoane.

Un poliedru reprezintă generalizarea în spațiul tridimensional a unui poligon din planul bidimensional: poliedrul este o regiune finită a spațiului a cărei suprafață de frontieră este compusă dintr-un număr finit de fețe poligonale plane. Suprafața de frontieră a unui poliedru conține trei tipuri de elemente geometrice: vârfurile (punctele), care sunt zero-dimensionale, muchiile (segmentele), care sunt unidimensionale și fețele (poligoanele), care sunt bidimensionale (fig. 2.4).

Suprafața de frontieră a unui poliedru este o colecție finită de fețe poligonale care se intersectează corect. Intersecția corectă a fețelor înseamnă că, pentru fiecare pereche de fețe ale obiectului, fețele sunt disjuncte, au în comun un singur vârf, sau au în comun două vârfuri și muchia care le unește.

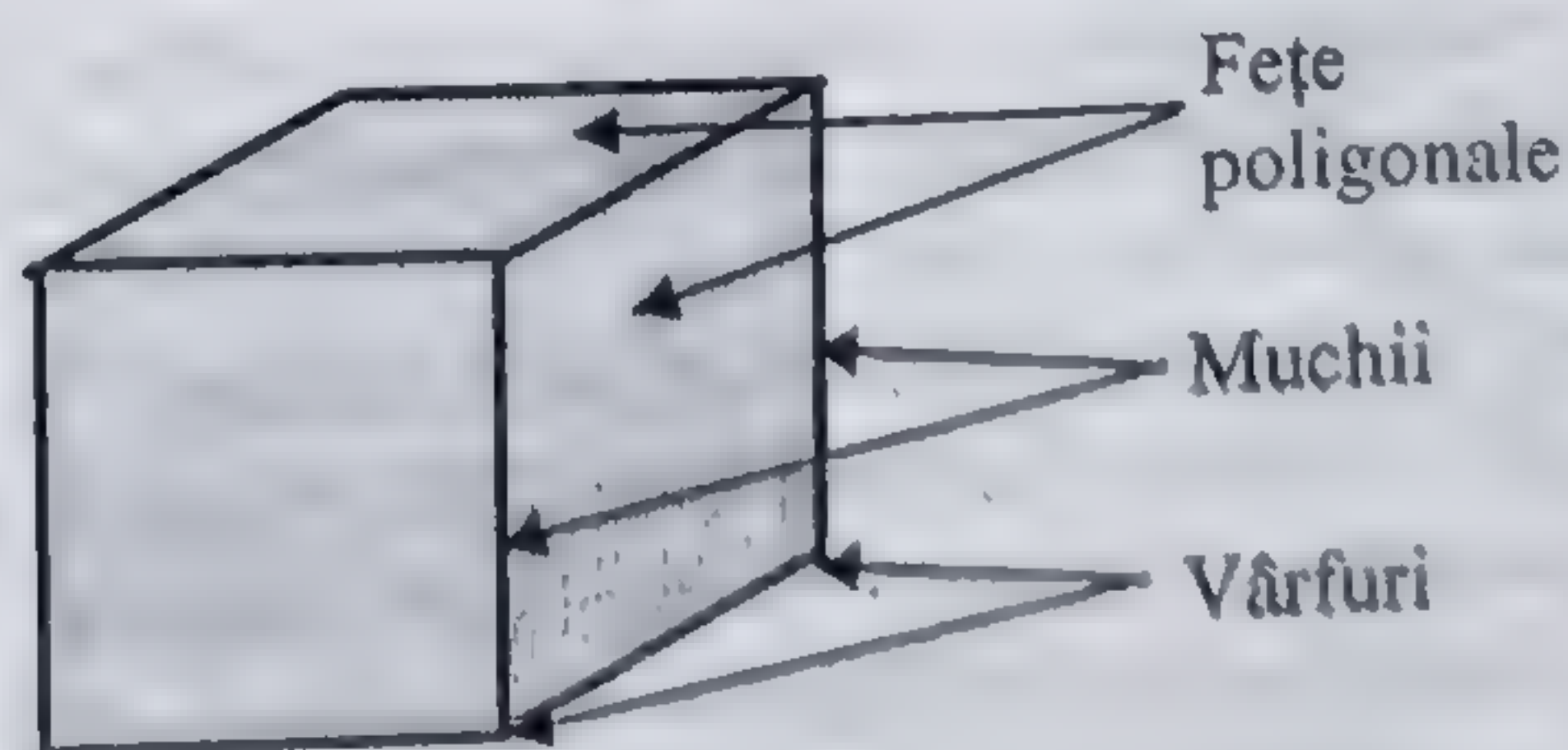


Fig. 2.4 Reprezentarea prin suprafața de frontieră a unui poliedru.

Din punct de vedere matematic, nu este imediat evident că un solid poate fi reprezentat univoc prin suprafața care îl mărginește. De aceea, este necesar să fie stabilite condițiile în care această reprezentare este permisă. Aceste condiții, numite



condiții de *construcție corectă*, se definesc pentru suprafețe de frontieră triangularizate. Triangularizarea unei suprafețe poliedrale se obține prin triangularizarea fiecărei fețe poligonale, astfel încât suprafața rezultată constă din vârfuri care sunt înconjurate de triunghiuri, fiecare pereche de triunghiuri fiind adiacente de-a lungul unei muchii. Laturile triunghiurilor adiacente unui vârf formează un circuit de segmente, numit link-ul vârfului (fig. 2.5).

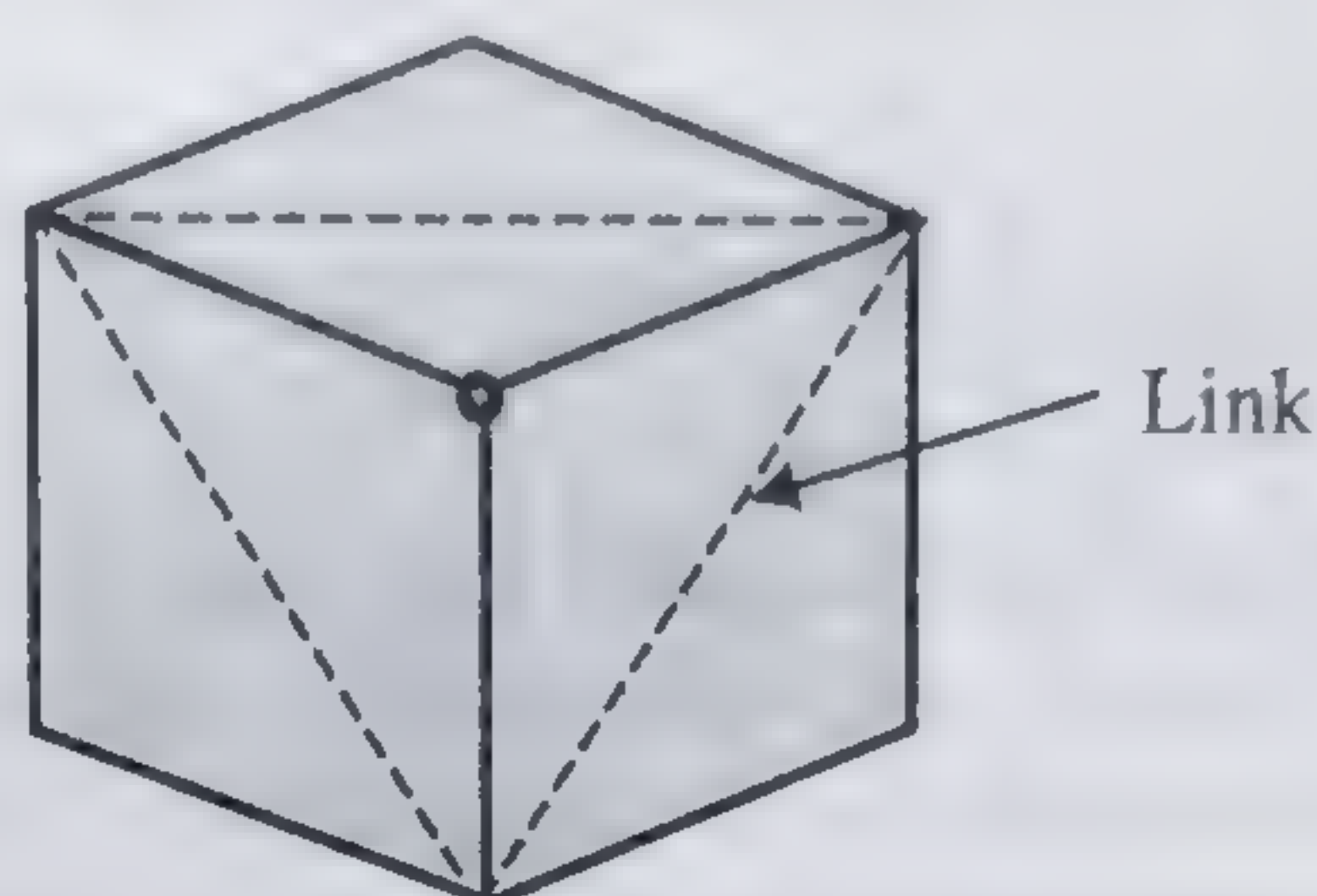


Fig. 2.5 Triangularizarea suprafeței de frontieră.  
Link-ul unui vârf al suprafeței.

O suprafață de frontieră construită corect îndeplinește următoarele condiții:

- Linkul fiecărui vârf al suprafeței triangularizate este complet, adică formează un circuit închis, nu neapărat planar.
- Triunghiurile suprafeței triangularizate sunt orientate consistent.

Închiderea link-ului fiecărui vârf asigură proprietățile suprafeței de a fi *închisă* și *conectată*. Proprietate de închidere înseamnă că suprafața nu are un sfârșit. Proprietatea de conectivitate înseamnă că există cel puțin o cale de la un triunghi la altul aparținând aceleiași suprafețe de frontieră, care poate fi parcursă prin traversarea muchiilor adiacente. Dacă o suprafață nu este închisă sau nu este conectată, prin triangularizarea suprafeței nu se obțin link-uri închise [Kal89].

Proprietate de construcție corectă a suprafeței de frontieră a unui solid se poate verifica folosind condiția de *orientare consistentă* a suprafeței, formulată de legea lui Moebius: O suprafață închisă este orientată consistent dacă la traversarea triunghiurilor sale (rezultate dintr-o triangularizare arbitrară), într-o direcție unică (de exemplu, în direcția inversă acelor de ceasornic), fiecare muchie este traversată de două ori, în direcții opuse (fig. 2.6).

Generalizarea în spațiul tridimensional a teoremei lui Jordan spune că orice suprafață de frontieră închisă, consistent orientată, împarte spațiul în două părți: o parte interioară suprafeței, care este o regiune limitată, și o parte exterioară suprafeței, care este o regiune nelimitată.

Din punct de vedere geometric, orientarea consistentă se verifică prin direcția normalelor la fețele obiectului: dacă normalele fețelor sunt îndreptate către aceeași regiune a spațiului (fie toate îndreptate spre interior, fie toate îndreptate spre exterior), atunci suprafața are o orientare consistentă. Acest mod de verificare se referă la obiectele tridimensionale fără cavități, dar se poate extinde cu ușurință și la obiecte care prezintă cavități.



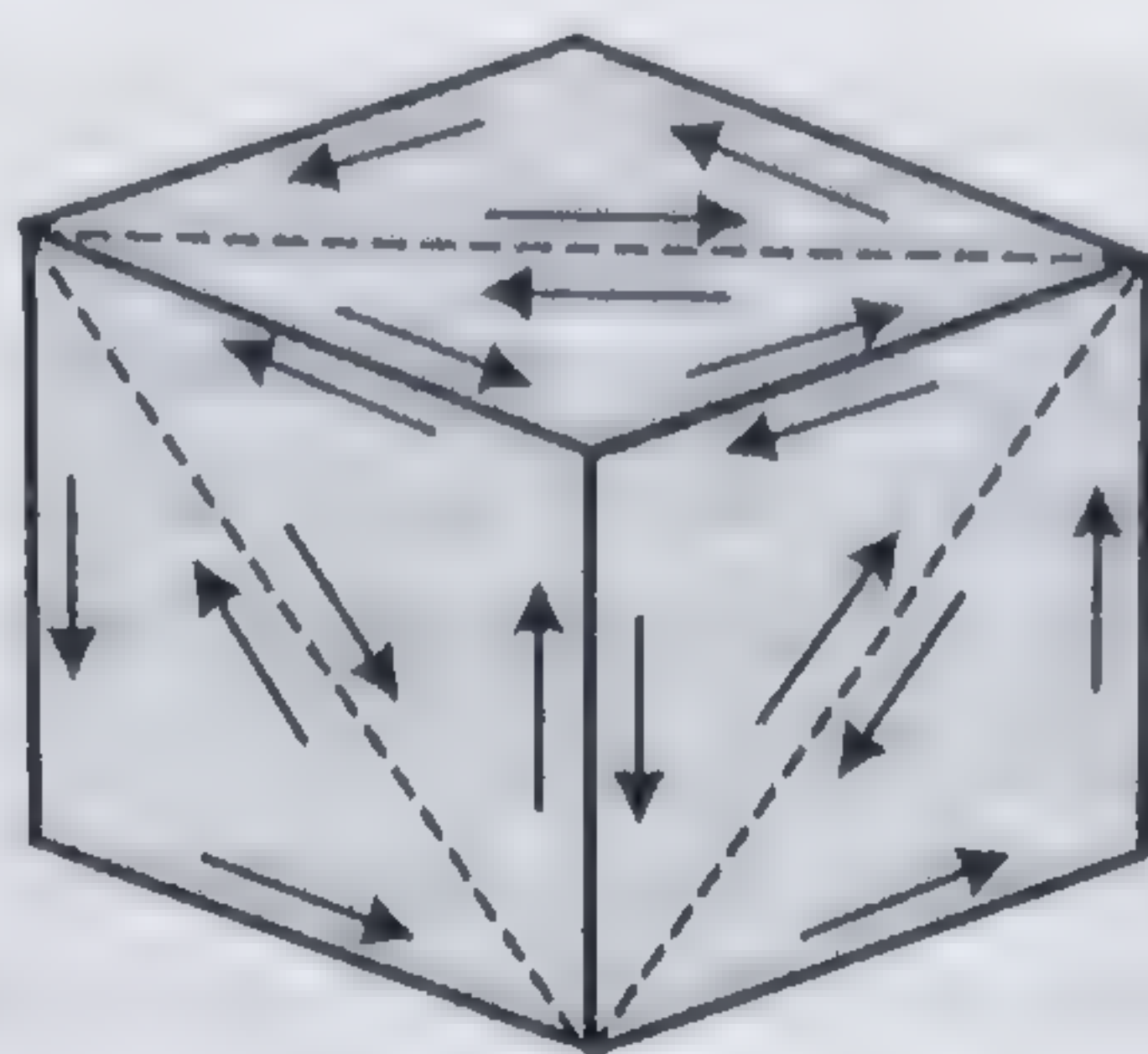


Fig. 2.6 Legea lui Mobius verifică orientarea consistentă suprafețelor triangularizate.

Teoretic, orientarea consistentă se verifică pentru fețele triangularizate ale suprafeței de frontieră, dar, prin extindere, se poate verifica orientarea consistentă folosind normalele la fețele poligonale, deoarece toate triunghiurile obținute prin triangularizarea unui poligon care reprezintă o față a unui obiect au aceeași orientare (fig. 2.7).

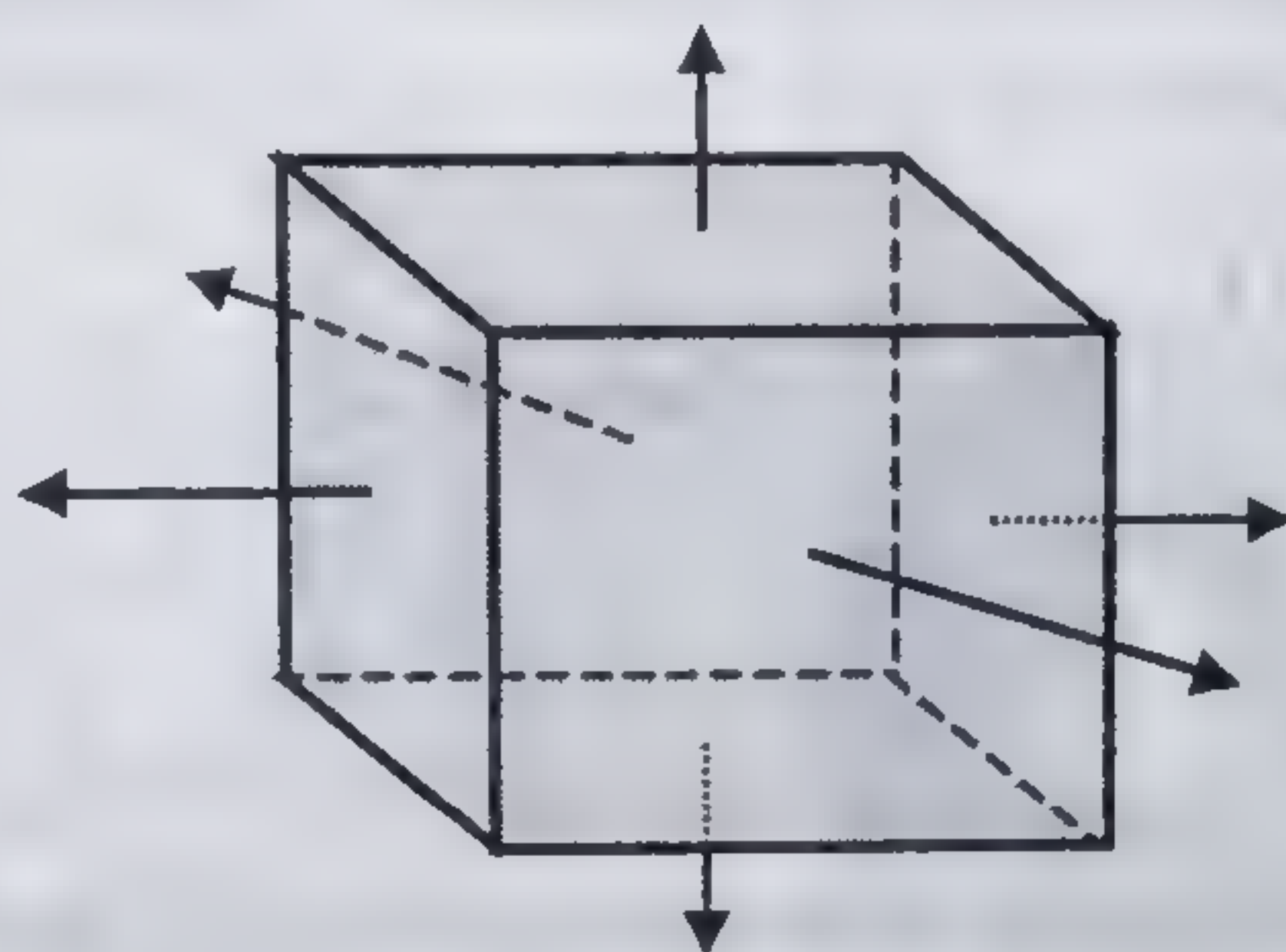


Fig. 2.7 Orientarea consistentă a fețelor obiectului.

Orientarea consistentă a fețelor poligonale ale obiectelor este o condiție de verificare a construcției corecte a suprafeței de frontieră și, în același timp, este folosită în operațiile de eliminare a suprafețelor ascunse în cursul redării obiectelor tridimensionale. De aceea, normalele la fețele poligonale se memorează în modelul obiectelor, împreună cu coordonatele vârfurilor.

### 2.1.3 IMPLEMENTAREA MODELULUI POLIGONAL

Reprezentarea prin rețea de poligoane a obiectelor se implementează printr-o listă de poligoane, care poate fi un vector sau o listă înlănțuită. Astfel, întreaga informație referitoare la forma unui obiect este compusă din liste de coordonate ale vârfurilor, la care se mai adaugă și alte informații geometrice necesare în redare (de exemplu, normalele la suprafețe).

Fără să se insiste acum asupra eficienței sau eleganței implementării (se va reveni ulterior), se poate considera că forma unui obiect modelat prin rețea de poligoane poate fi reprezentată ca un vector de fețe, fiecare față fiind un vector de vârfuri, fiecare vârf fiind un vector de trei coordonate în virgulă flotantă. De exemplu, un cub poate fi reprezentat astfel:



```

double CubeFaces[6][4][3] = {
    {{-1,-1,-1},          // fata 0
     { 1,-1,-1},
     { 1,-1, 1},
     {-1,-1, 1}},
    {{-1, 1, 1},          // fata 1
     { 1, 1, 1},
     { 1, 1,-1},
     {-1, 1,-1}},
    {{-1,-1, 1},          // fata 2
     { 1,-1, 1},
     { 1, 1, 1},
     {-1, 1, 1}},
    {{ 1,-1,-1},          // fata 3
     {-1,-1,-1},
     {-1, 1,-1},
     { 1, 1,-1}},
    {{ 1,-1, 1},          // fata 4
     { 1,-1,-1},
     { 1, 1,-1},
     { 1, 1, 1}},
    {{-1,-1, 1},          // fata 5
     {-1, 1, 1},
     {-1, 1,-1},
     {-1,-1,-1}}
};

```

O astfel de implementare este însă ineficientă deoarece fiecare vârf este prelucrat de trei ori, pentru fiecare față adiacentă acestuia. O implementare mult mai eficientă a modelului poligonal definește un vector cu toate vârfurile unui obiect, iar fiecare față se definește printr-un vector de indecși în vectorul de vârfuri. De exemplu, modelul unui cub folosind indecși pentru definirea fețelor arată astfel:

```

double CubeCoords[8][3]={
    {-1,-1, 1},
    { 1,-1, 1},
    { 1,-1,-1},
    {-1,-1,-1},
    {-1, 1, 1},
    { 1, 1, 1},
    { 1, 1,-1},
    {-1, 1,-1}
};
int CubeIndexFace[6][4]={
    {3, 2, 1, 0},
    {4, 5, 6, 7},
    {0, 1, 5, 4},
    {2, 3, 7, 6},
    {1, 2, 6, 5},
    {0, 4, 7, 3}
};

```



Atât în cursul modelării, cât și în cursul redării imaginii, modelul obiectelor se reprezintă mai complex, conținând și alte informații geometrice și atribute de aspect. Astfel, culoarea (mai general spus, materialul) este o informație care se asignează fiecărei fețe sau fiecărui vârf al obiectului. În general, modelul poligonal cu reprezentarea prin indecși a fețelor poligonale este implementat orientat pe obiecte, folosind mai multe clase (de bază și derivate) care permit încapsularea tuturor informațiilor necesare pentru redarea obiectelor.

O rețea de poligoane poate să reprezinte fie suprafața de frontieră a unui solid, fie o suprafață deschisă, necesară în anumite situații de modelare, cum este suprafața terenului simulat într-o scenă virtuală. Deoarece se modelează o zonă geografică limitată și nu se explorează scena astfel ca să fie văzută marginea “pământului”, se poate folosi o suprafață poligonală deschisă pentru reprezentarea terenului.

Reprezentarea obișnuită în grafica din scenele virtuale este aceea în care fiecare poligon este reprezentat ca o suprafață (poligoane “pline”), dar în proiectările grafice (în mecanică, construcții, etc) se folosește și reprezentarea prin contur a poligoanelor, numită reprezentare “cadru de sârmă” (*wireframe*) (fig. 2.8).

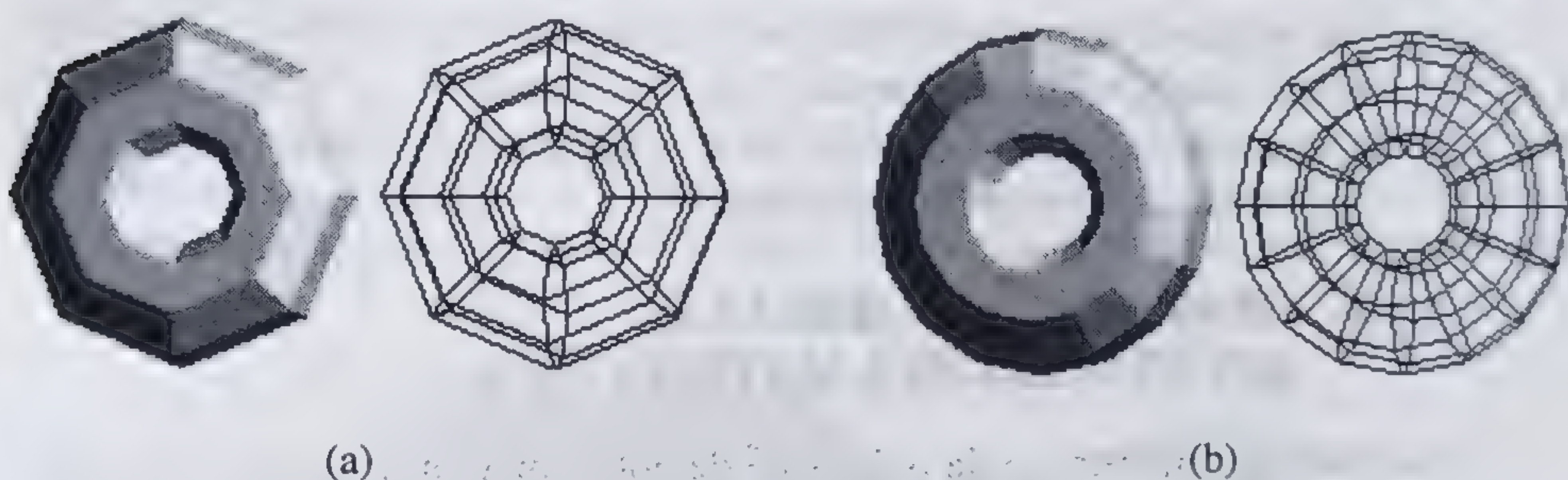


Fig. 2.8 Reprezentarea “plină” (*filled*) și “cadru de sârmă” (*wireframe*) a obiectelor.

Precizia de reprezentare a modelului, adică diferența dintre suprafața obiectului și fețele poligonale prin care este aproximat, este un parametru important de modelare. În general, cu cât numărul de poligoane prin care se aproximează suprafața obiectului este mai mare, cu atât precizia de reprezentare este mai bună. Se observă diferența dintre imaginea unui obiect (tor) reprezentat prin 64 de poligoane în fig. 2.8(a) și prin 128 de poligoane în fig. 2.8 (b).

Folosirea unui număr mare de poligoane pentru reprezentarea obiectelor conduce la un volum imens de date în modelarea scenelor virtuale, care implică cerințe corespunzătoare de memorare și de capacitate de calcul. Oricât de mult ar crește capacitatea de prelucrare în sistemele grafice, prin folosirea paralelismului și prin implementarea hardware a algoritmilor de prelucrare, performanțe de generare interactivă în timp real a imaginilor se pot obține numai dacă obiectele scenei se modelează în mod corespunzător. Metodele de aproximare eficientă a modelelor sunt cunoscute sub numele de *simplificarea* (sau *rafinarea*) *datelor*.



Tehnica de simplificare directă, prin reducerea uniformă a numărului de poligoane de reprezentare a obiectului, este inacceptabilă, deoarece nu se poate controla precizia de reprezentare. Pentru păstrarea unei precizii de reprezentare constante, se folosește modelarea adaptivă, în care dimensiunea fețelor poligonale variază în funcție de curbura suprafeței; în părțile cu curbură pronunțată sunt introduse mai multe poligoane pe unitatea de suprafață.

O altă metodă de simplificare a datelor este modelarea cu nivele de detaliu multiple a obiectelor (*levels of detail*- LOD), în care modelul unui obiect este compus dintr-o succesiune de reprezentări, fiecare cu o precizie diferită. În cursul generării imaginilor, se selectează nivelul de detaliu corespunzător, în funcție de poziția obiectului față de punctul de observare. Cu cât obiectul este mai depărtat, se selectează o reprezentare cu o precizie mai scăzută a obiectului. Această metodă este posibilă datorită folosirii proiecției perspective în generarea imaginilor și va fi înțeleasă mai ușor după parcurgerea capitolului următor privind transformările geometrice.

Modelul poligonal al unui obiect se poate genera prin mai multe metode, în funcție de tipul obiectului și de aplicația grafică în care este folosit modelul respectiv. Se poate folosi una din următoarele metode de modelare poligonală:

- Generarea modelului din descrierea matematică a obiectului.
- Generarea modelului obiectului prin baleiere spațială.
- Generarea modelului pornind de la o mulțime de puncte care aparțin suprafeței de frontieră a obiectului.

#### 2.1.4 GENERAREA MODELULUI DIN DESCRIEREA MATEMATICĂ

Se poate genera rețeaua de poligoane de aproximare a obiectelor care au o descriere matematică cunoscută. De exemplu, ecuațiile unor suprafețe quadrice:

- Elipsoid:

$$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0 \quad (2.1)$$

unde  $a, b, c$  sunt semiaxele elipselor.

- Hiperboloid:

$$x^2/a^2 + y^2/b^2 - z^2/c^2 - 1 = 0 \quad \text{și} \quad (2.2)$$

$$x^2/a^2 + y^2/b^2 - z^2/c^2 + 1 = 0$$

- Paraboloid eliptic:

$$x^2/a^2 + y^2/b^2 = z \quad (2.3)$$

Suprafața se intersectează mai întâi cu un număr  $n$  de plane perpendiculare pe axa  $Oz$ , de ecuații  $z = n \Delta z$ , pentru  $n = -k, -k+1, \dots, -1, 0, 1, 2, \dots, k-1, k$ . Se obțin  $n$  elipse (paralele) și pe fiecare elipsă se eșantionează  $m$



puncte echidistante (pe meridiane), obținându-se  $(n-1) \times m$  poligoane care aproximează suprafața de frontieră a obiectului respectiv.

Aceste suprafețe se pot obține și prin rotația unei curbe în jurul unei axe de rotație. De exemplu, suprafața elipsoidului se obține prin rotația în jurul axei  $z$  a elipsei:

$$\begin{aligned} x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 &= 0, \\ y &= 0. \end{aligned} \quad (2.4)$$

Prin rotația unei curbe în jurul unei axe se pot obține obiecte tridimensionale mai variate, în funcție de forma curbei care se rotește. De exemplu, un tor se obține prin rotația unui cerc în jurul unei axe paralele cu planul cercului. Suprafețele astfel obținute se numesc suprafețe de rotație.

### 2.1.5 GENERAREA MODELULUI PRIN BALEIERE SPAȚIALĂ

Se pot genera obiecte tridimensionale prin deplasarea (*sweeping*) unei suprafețe generatoare de-a lungul unei curbe oarecare. Dacă se variază forma și orientarea suprafeței generatoare în cursul deplasării, se pot obține obiecte variate, în funcție de forma curbei și de orientarea, forma și variația formei suprafeței generatoare. Prin această metodă se pot obține atât formele regulate descrise mai sus (elipsoid, hiperboloid, paraboloid eliptic, tor) cât și alte obiecte numite solide ductibile sau extrudate (*ducted solids*) sau cilindri generalizați (*generalized cylinders*) (fig. 2.9).

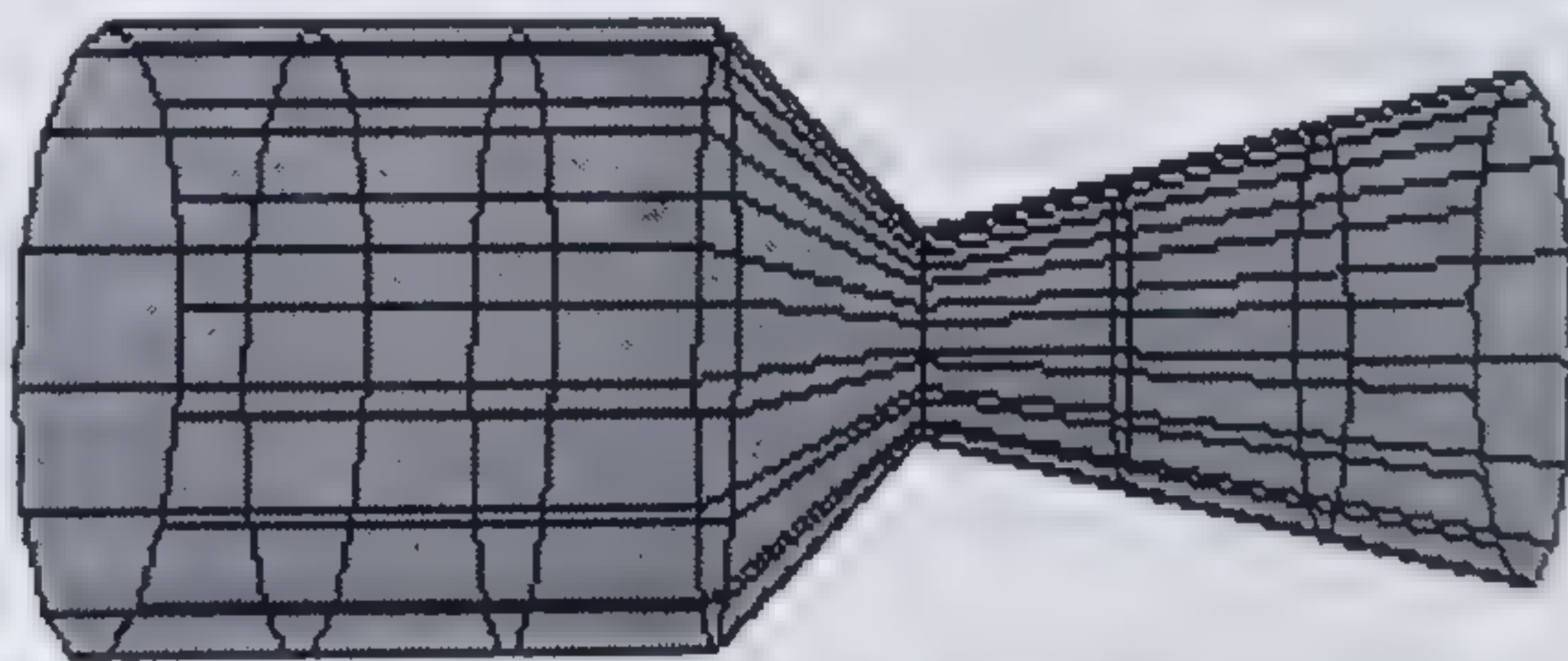


Fig. 2.9 Obiect poligonal modelat prin deplasarea unui cerc cu diametru variabil de-a lungul unei linii drepte.

Pentru definirea deplasării unei suprafețe de-a lungul unei curbe, este necesar să se definească intervalul curbei pe care are loc deplasarea și modul în care se divide intervalul parcurs. Împărțirea intervalului în distanțe egale nu dă rezultate bune, deoarece punctele obținute nu vor fi egal distribuite pe suprafața obiectului. De aceea este necesară divizarea intervalului în funcție de curbura curbei. Dacă curbura este pronunțată, se aleg subdiviziuni mai mici ale intervalului, iar pentru curburi mai reduse se aleg subdiviziuni mai mari ale intervalului.



### 2.1.6 GENERAREA MODELULUI PE BAZA UNEI MULȚIMI DE PUNCTE CARE APARTIN SUPRAFETEI DE FRONTIERĂ A OBIECTULUI

O metodă mai generală de modelare a obiectelor tridimensionale se bazează pe cunoașterea unei mulțimi de puncte distribuite uniform sau neuniform pe suprafața obiectelor. Această metodă implică mai întâi obținerea colecției de puncte și apoi construirea rețelei de poligoane care să aproximeze obiectul cu precizia dorită. Dacă metodele prezentate mai înainte pot fi utilizate pentru o categorie restrânsă de obiecte, în schimb, metoda generării modelului pornind de la o mulțime de puncte ale suprafeței acestuia poate fi aplicată pentru cele mai variate obiecte: clădiri, vehicule, plante, animale, elemente anatomice, teren, etc.

Mulțimea de puncte de pe suprafața obiectului se determină diferit, în funcție de modul în care este cunoscut sau reprezentat obiectul real:

- Pentru obiectele care se proiectează într-un sistem de proiectare asistat de calculator – CAD (*Computer Aided Design*), informațiile de formă necesare în generearea modelului se pot obține din reprezentarea proiectului. De exemplu, clădiri, obiecte mecanice, vehicule pot fi proiectate în AutoCad și coordonatele vârfurilor suprafețelor sunt folosite pentru reprezentarea prin rețea poligonală a obiectului.
- Pentru obiectele reale, sau machete ale acestora, se poate folosi un scanner 3D cu laser. Obiectul real (sau macheta acestuia) este plasat pe o masă rotativă în direcția de emisie a unei raze laser. La fiecare rotație completă a mesei se obține o colecție de puncte pe un contur al obiectului, prin măsurarea distanței la suprafața obiectului. După fiecare rotație, se deplasează masa în sus sau în jos, astfel că se obține o colecție de contururi ale obiectului. Toate aceste puncte de pe suprafața obiectului sunt folosite pentru crearea modelului poligonal.
- Pentru modelarea terenului se folosesc hărți digitale care furnizează altitudinile terenului într-o grilă uniformă de puncte, sau contururi de nivel, care unesc puncte cu altitudine constantă.

Fiind cunoscută o mulțime de puncte  $V$  care caracterizează un obiect tridimensional, volumul  $\Omega \subset \mathbb{R}^3$  acoperit de aceste puncte se numește domeniul mulțimii  $V$ .  $\Omega$  este un poliedru, convex sau neconvex, iar punctele mulțimii  $V$  pot fi dispuse regulat sau neregulat în domeniul  $\Omega$ . Cazul cel mai frecvent întâlnit în modelare este acela în care punctele mulțimii  $V$  sunt distribuite neregulat în spațiu și aproximarea obiectului se realizează printr-o rețea tridimensională de poligoane  $\Gamma$ , care aproximează liniar pe porțiuni obiectul dat. Rețeaua  $\Gamma$  trebuie să fie construită pornind de la mulțimea  $V$  de puncte în spațiu și de la cerințele de precizie de aproximare a obiectului. Construirea modelului poligonal al unui obiect cunoscut printr-o mulțime de puncte de pe suprafața sa se poate realiza prin mai multe metode, dintre care cele mai folosite sunt triangularizarea (*triangulation*) și transformata wavelet [Gross96]. În continuare sunt prezentate proprietățile și principiile de realizare a triangularizării.



### 2.1.7 TRIANGULARIZAREA UNEI MULȚIMI DE PUNCTE

Triangularizarea în două sau trei dimensiuni a fost intens studiată, începând cu Dirichlet, Voronoi și continuând cu Delaunay. Un număr mare de cărți și articole tratează proprietățile triangularizării și algoritmi de construcție a modelelor prin triangularizare [Aur91], [Rour93]. Dintre diferitele modalități de triangularizare, triangularizarea Delaunay are proprietatea că unghiurile minime ale triunghiurilor generate sunt maxime față de toate celelalte triangularizări. Această proprietate este deosebit de utilă în grafică, unde suprafețele ascuțite (cu unghiuri mici) provoacă un zgomot (zgomot de aliasing), care crește cu cât scade dimensiunea primitivelor geometrice (triunghiurile).

Triangularizarea Delaunay este o structură duală uneia din cele mai importante structuri din geometrie, numită diagrama Voronoi. Conceptul de diagramă Voronoi are peste un secol vechime, fiind descris de Dirichlet în 1850 și de Voronoi în 1908. Diagrama Voronoi are numeroase aplicații în diferite domenii ale științei și tehnicii (codare, robotică, cristalografie, etc).

#### 2.1.7.1 Diagrama Voronoi în plan

Fie o mulțime de puncte  $P = \{p_1, p_2, \dots, p_n\}$  în spațiul euclidian bidimensional. Aceste puncte se numesc locații (*sites*). Se partitionează planul bidimensional prin asignarea fiecărui punct din plan celei mai apropiate locații,  $p_i$ . Toate aceste puncte formează *regiunea Voronoi a locației*  $R(p_i)$ . (Regiunea Voronoi se mai numește și domeniul Dirichlet, regiunea Wigner-Seitz sau polinomul Tiessen; această regiune nu este, însă, poligon în sensul definiției date în paragraful precedent, deoarece poate fi nemărginită). Regiunea  $R(p_i)$  constă din mulțimea tuturor punctelor cel puțin la fel de apropiate de  $p_i$  ca de oricare altă locație:

$$R(p_i) = \{x: |p_i - x| \leq |p_j - x|, \forall j \neq i\} \quad (2.5)$$

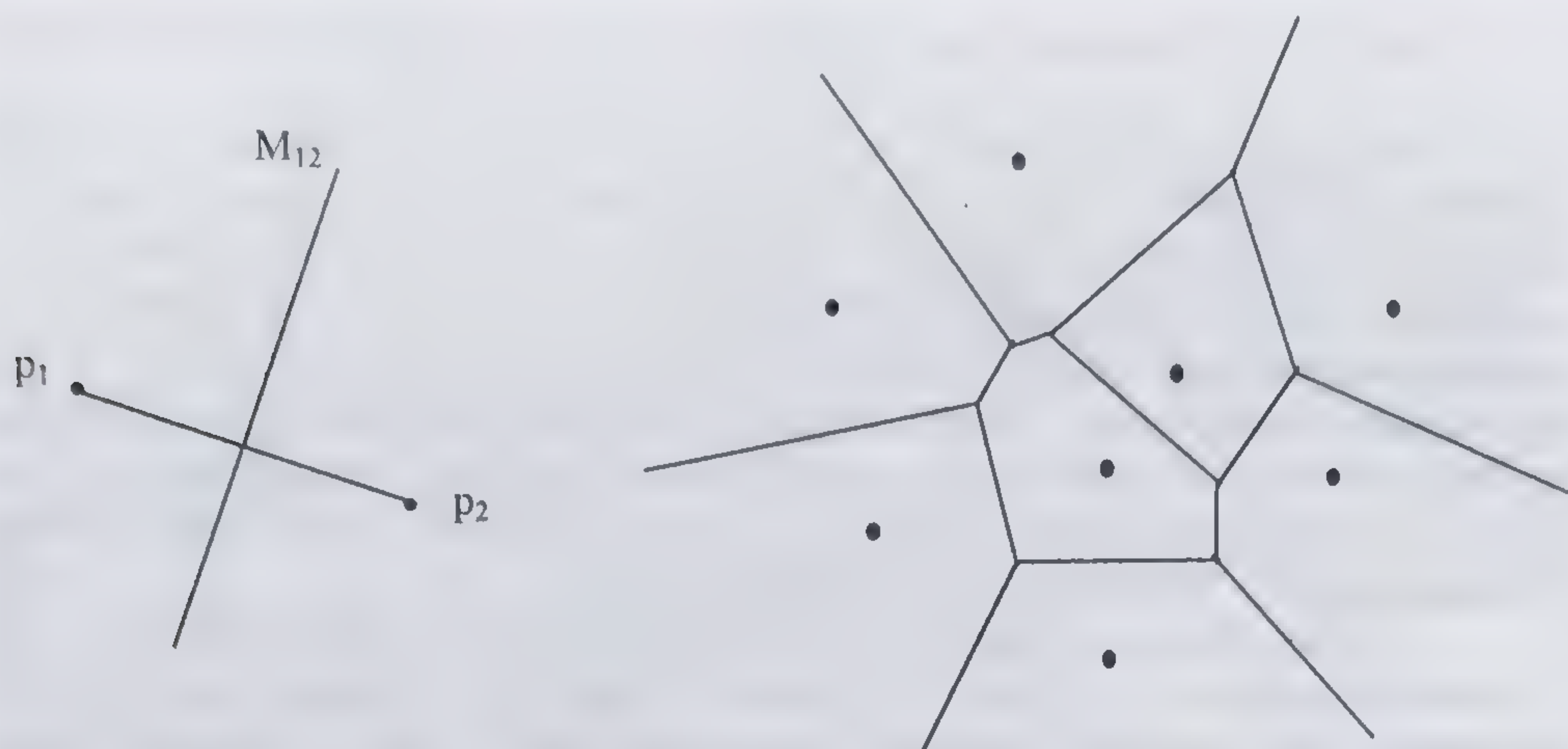
Unele puncte din plan nu au o singură locație cea mai apropiată (vecină). Mulțimea punctelor care au mai mult de o locație vecină formează diagrama Voronoi  $V(P)$ . Pentru două puncte în plan  $p_1$  și  $p_2$ , mediatoarea segmentului  $p_1p_2$ , notată  $M_{12} = M(p_1, p_2)$ , împarte planul în două semiplane, unul asignat punctului  $p_1$ , celălalt asignat punctului  $p_2$  (fig. 2.10(a)). Pentru mai mult de două puncte în plan, se consideră mediatoarea  $M_{ij}$  a fiecărei perechi de puncte  $p_i$  și  $p_j$ , și  $H(p_i, p_j)$  semiplanul închis de dreapta  $M_{ij}$  care conține locația  $p_i$ .

Dat fiind că regiunea Voronoi a unei locații  $R(p_i)$  este mulțimea tuturor punctelor mai apropiate de  $p_i$  decât de oricare altă locație, înseamnă că ea conține toate punctele mai apropiate de  $p_i$  decât de  $p_1$  și mai apropiate de  $p_i$  decât de  $p_2$  și mai apropiate de  $p_i$  decât de  $p_3$  și așa mai departe. Se poate scrie următoarea ecuație pentru  $R(p_i)$ :

$$R(p_i) = \bigcap_{j \neq i} H(p_i, p_j) \quad (2.6)$$

Intersecția semiplanelor  $H(p_i, p_j)$  se calculează pentru orice  $i \neq j$  și rezultă diagrama Voronoi a mulțimii de puncte date (fig. 2.10(b)).



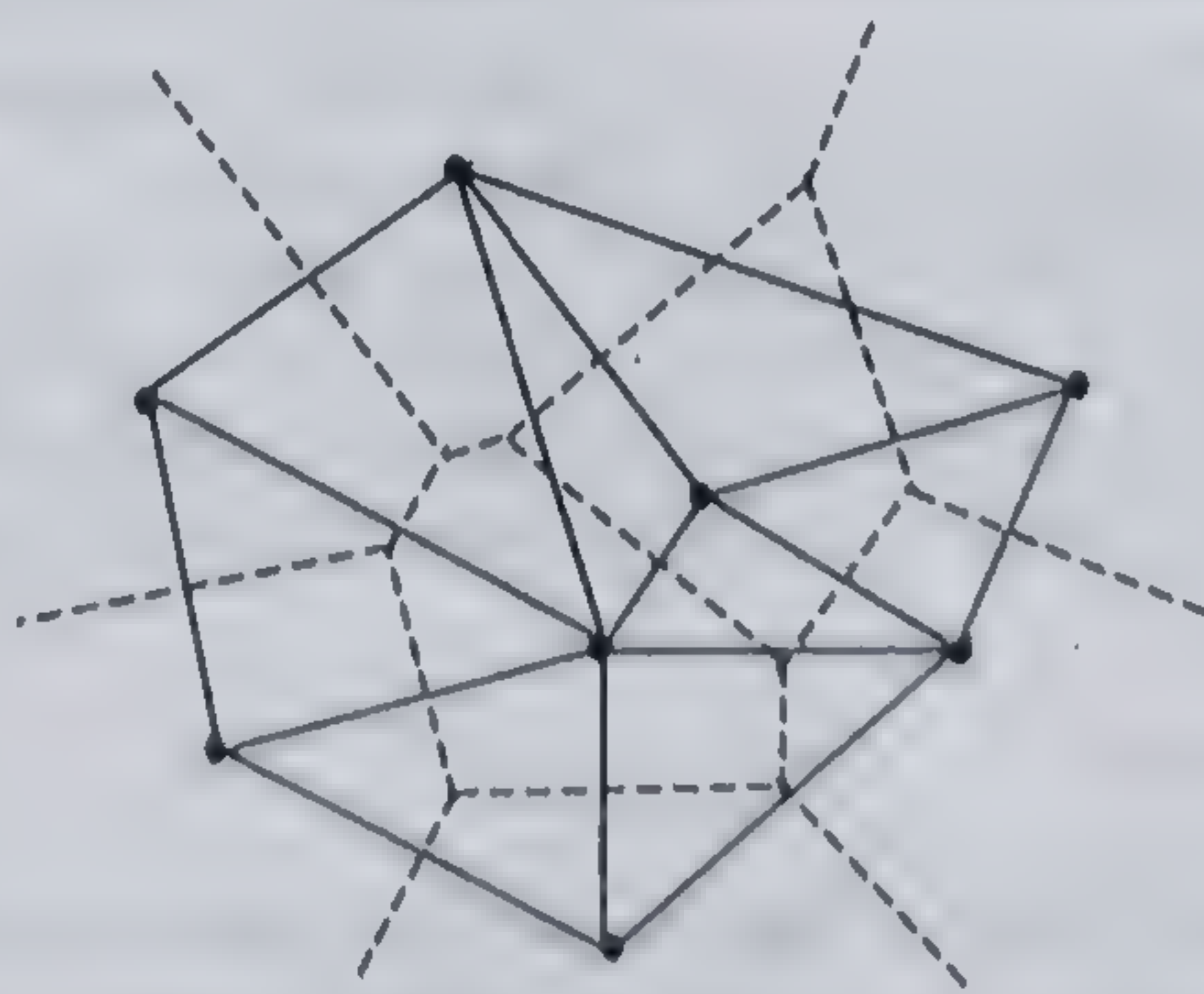


**Fig. 2.10** Diagrame Voronoi.  
(a) Pentru două locații (b) Pentru  $n = 6$  locații.

Diagrama Voronoi a  $n$  puncte conține  $n$  regiuni Voronoi și toate sunt regiuni convexe, dat fiind că intersecția dintre oricâte semiplane este o mulțime convexă de puncte în plan. Dacă regiunile sunt limitate, ele sunt poligoane convexe. În diagrama Voronoi, muchiile se numesc muchii Voronoi, iar intersecțiile între muchii definesc vârfurile Voronoi. Oricare punct de pe muchiile Voronoi are două locații vecine, iar oricare vârf are cel puțin trei locații vecine.

#### 2.1.7.2 Triangularizarea Delaunay în plan

Se definește graful dual al unei diagrame Voronoi  $V(P)$  drept graful  $G$  care se construiește în felul următor: nodurile grafului sunt locațiile diagramei  $V(P)$  și două noduri în graf sunt conectate între ele dacă regiunile Voronoi corespunzătoare locațiilor acestora au o muchie Voronoi comună. În 1934 Delaunay a demonstrat că, dacă muchiile grafului dual al diagramei Voronoi se reprezintă prin segmente de dreaptă și nu există patru locații conciclice, atunci se obține o triangularizare plană a locațiilor  $P$ . Această triangularizare se numește triangularizare Delaunay, notată  $D(P)$  (fig. 2.11). Cazul cu patru locații conciclice este studiat și tratat separat.



**Fig. 2.11** Triangularizarea Delaunay este graful dual al diagramei Voronoi.



Diagrama Voronoi și triangularizarea Delaunay sunt structuri duale, care conțin aceleași informații, dar reprezentate în forme diferite. Câteva din proprietățile acestor structuri sunt:

- Fiecare triunghi din  $D(P)$  corespunde unui vârf în  $V(P)$ .
- Fiecare muchie din  $D(P)$  corespunde unei muchii în  $V(P)$ .
- Fiecare nod (vârf) din  $D(P)$  corespunde unei locații în  $V(P)$ .
- Frontiera lui  $D(P)$  reprezintă acoperirea convexă a punctelor.
- În interiorul fiecărui triunghi din  $D(P)$  nu există nici o locație.
- Fiecare regiune Voronoi  $V(p_i)$  este convexă.
- Regiunea  $V(p_i)$  este nelimitată dacă și numai dacă  $p_i$  se află pe frontiera (acoperirea convexă) triangularizării Delaunay.
- Dacă  $v$  este un vârf Voronoi la intersecția regiunilor  $V(p_1)$ ,  $V(p_2)$ ,  $V(p_3)$ , atunci  $v$  este centrul cercului  $C(v)$ , determinat de  $p_1$ ,  $p_2$ ,  $p_3$ . Această proprietate se generalizează pentru vârfuri de orice grad.
- Cercul  $C(v)$ , circumscris triunghiului Delaunay corespunzător vârfului  $v$ , nu conține nici o locație în interiorul lui.
- Dacă  $p_j$  este cel mai apropiat vecin al lui  $p_i$ , atunci  $(p_i, p_j)$  este o muchie în  $D(P)$ .
- Dacă există un cerc care trece prin  $p_i$  și  $p_j$  și nu conține nici o locație în interiorul său, atunci  $(p_i, p_j)$  este o muchie în  $D(P)$ . Inversa acestei proprietăți este: pentru fiecare muchie Delaunay există un cerc vid (care nu conține nici o locație în interiorul său).

Demonstrația acestor proprietăți se poate găsi în mai multe lucrări [Rour93], [Prep85]. Ultima proprietate este cunoscută sub numele de *proprietatea cercului vid* (*empty circle property*) și este una din metodele de triangularizare Delaunay. Numeroasele aplicații și frumusețea structurilor Voronoi și Delaunay au condus la cercetări intense pentru elaborarea algoritmilor de generare a acestora. În grafică interesează cel mai mult triangularizarea unei mulțimi de puncte. Triangularizarea se poate construi direct sau prin intermediul diagramei Voronoi, care, uneori, se obține mai ușor. Pornind de la diagrama Voronoi, triangularizarea Delaunay se obține construind câte o muchie pentru fiecare pereche de vârfuri Voronoi vecine (care au o muchie Voronoi comună).

**Metoda cercului vid.** Metoda cea mai directă de construire a triangularizării Delaunay a unei mulțimi  $P$  de puncte este cea bazată pe proprietatea cercului vid: pentru fiecare triplet  $(p_i, p_j, p_k)$  se testează incluziunea oricărei alte locații: dacă cercul respectiv nu conține nici o locație, atunci muchiile triunghiului  $(p_i, p_j, p_k)$  aparțin triangularizării  $D(P)$ . Acest algoritm este extrem de simplu de realizat (conține trei bucle imbricate în care se testează incluziunea unui punct într-un cerc), dar este unul dintre cei mai lenți algoritmi de triangularizare, având o complexitate în  $O(n^3)$  și nu poate fi folosit decât pentru număr redus de puncte.

**Algoritmul de triangularizare divide-and-conquer.** Algoritmul cel mai rapid de construire a diagramei Voronoi, din care se poate obține imediat



triangularizarea Delaunay, este algoritmul de tip împarte-și-cucerește (*divide and conquer*) definit de Shamos și Hoey în 1975, cu o complexitate în  $O(n \log n)$ . Acest algoritm este însă deosebit de dificil de implementat și de aceea sunt algoritmi mai puțin eficienți, dar mai simpli.

**Algoritmul incremental.** Unul dintre cei mai populari algoritmi de triangularizare este algoritmul incremental, care are o complexitate în  $O(n^2)$ . Dacă triangularizarea Delaunay  $D$  a unei mulțimi de  $k$  puncte, se poate construi triangularizarea  $D'$  pentru mulțimea de puncte inițială la care se adaugă un nou punct  $p$ . Pentru inserarea unui nou punct  $p$  într-o triangularizare se determină întâi regiunea  $R$  a noului punct, formată din mulțimea triunghiurilor al căror circumscris conține punctul  $p$  (fig. 2.12).

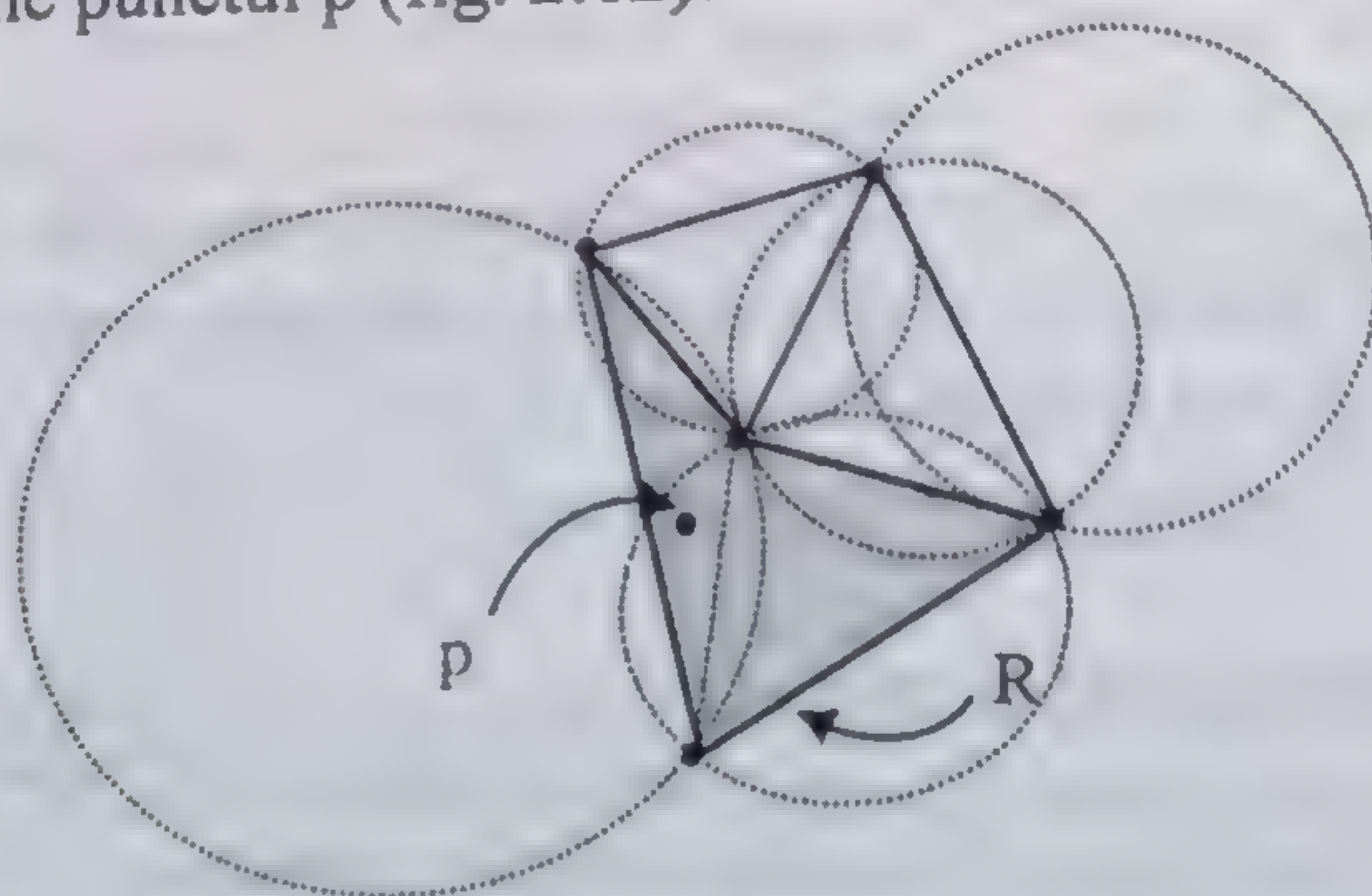


Fig. 2.12 Algoritmul incremental de triangularizare Delaunay.

Regiunea  $R$  are proprietatea că nu conține nici un alt punct în interiorul ei, cu excepția lui  $p$ . Din această regiune se reține frontiera prin eliminarea muchiilor interne. După eliminarea muchiilor interne, toate vârfurile regiunii vizibile din punctul  $p$  și triangularizarea  $D'$  se obține prin unirea punctului  $p$  cu fiecare din vârfurile regiunii  $R$ .

### 2.1.7.3 Triangularizarea Delaunay în spațiul tridimensional

Se consideră o mulțime de puncte în spațiu (locații),  $(x_i, y_i, z_i)$ ,  $i = 1, \dots, n$ . Triangularizarea Delaunay în spațiu înseamnă umplerea domeniului acoperit de mulțimi de puncte cu tetraedre, astfel încât vârfurile tetraedrelor să fie puncte din mulțimea dată și sfera circumscrisă fiecărui tetraedru să nu conțină nici o altă locație în interiorul ei. Această condiție este extinderea în spațiul tridimensional a condiției cercului vid din triangularizarea Delaunay în plan.

Algoritmii de construire a triangularizării Delaunay în spațiul tridimensional sunt destul de complicați, necesitând testarea a numeroase situații particulare care apar datorită capacității limitate de reprezentare a valorilor coordonatelor [Fang95]. Rezultatul triangularizării Delaunay este o rețea (de multe ori neregulată) de poligoane în spațiu, prin care se aproximează suprafața de frontieră a obiectului respectiv. Aparatura modernă de scanare 3D include algoritmi de triangularizare tridimensională, astfel că utilizatorul primește direct modelul poligonal al obiectului scanat.



triangularizarea Delaunay, este algoritmul de tip împarte-și-cucerește (*divide-and-conquer*) definit de Shamos și Hoey în 1975, cu o complexitate în  $O(n \log n)$ . Acest algoritm este însă deosebit de dificil de implementat și de aceea sunt folosiți algoritmi mai puțin eficienți, dar mai simpli.

**Algoritmul incremental.** Unul dintre cei mai populari algoritmi de triangularizare este algoritmul incremental, care are o complexitate în  $O(n^2)$ . Fiind dată triangularizarea Delaunay  $D$  a unei mulțimi de  $k$  puncte, se poate construi triangularizarea  $D'$  pentru mulțimea de puncte inițială la care se adaugă încă un punct  $p$ . Pentru inserarea unui nou punct  $p$  într-o triangularizare se determină mai întâi regiunea  $R$  a noului punct, formată din mulțimea triunghiurilor al cărui cerc circumscris conține punctul  $p$  (fig. 2.12).

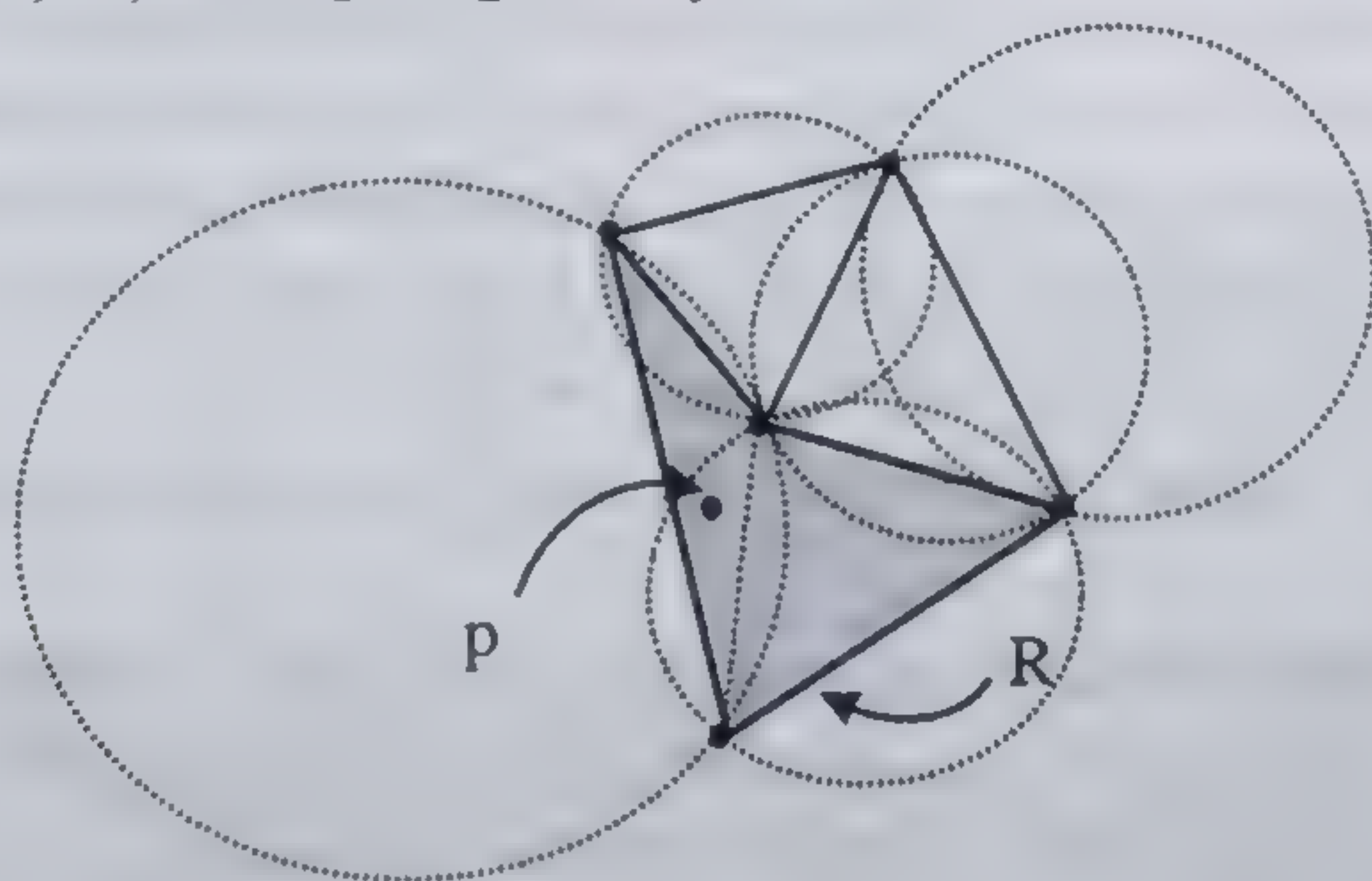


Fig. 2.12 Algoritmul incremental de triangularizare Delaunay.

Regiunea  $R$  are proprietatea că nu conține nici un alt punct în interiorul ei, cu excepția lui  $p$ . Din această regiune se reține frontiera prin eliminarea tuturor muchiilor interne. După eliminarea muchiilor interne, toate vârfurile regiunii sunt vizibile din punctul  $p$  și triangularizarea  $D'$  se obține prin unirea punctului  $p$  cu fiecare din vârfurile regiunii  $R$ .

### 2.1.7.3 Triangularizarea Delaunay în spațiul tridimensional

Se consideră o mulțime de puncte în spațiu (locații),  $(x_i, y_i, z_i)$ ,  $i = 1, \dots, n$ . Triangularizarea Delaunay în spațiu înseamnă umplerea domeniului acestei mulțimi de puncte cu tetraedre, astfel încât vârfurile tetraedrelor să fie puncte din mulțimea dată și sfera circumscrisă fiecărui tetraedru să nu conțină nici o altă locație în interiorul ei. Această condiție este extinderea în spațiul tridimensional a condiției cercului vid din triangularizarea Delaunay în plan.

Algoritmii de construire a triangularizării Delaunay în spațiul tridimensional sunt destul de complicați, necesitând testarea a numeroase situații particulare care apar datorită capacității limitate de reprezentare a valorilor coordonatelor [Fang95]. Rezultatul triangularizării Delaunay este o rețea (de cele mai multe ori neregulată) de poligoane în spațiu, prin care se aproximează suprafața de frontieră a obiectului respectiv. Aparatura modernă de scanare 3D are înglobați algoritmi de triangularizare tridimensională, astfel că utilizatorul primește direct modelul poligonal al obiectului scanat.



#### 2.1.7.4 Triangularizarea suprafețelor de teren

Un caz special de triangularizare tridimensională este triangularizarea suprafețelor de teren. În aplicațiile care necesită reprezentarea terenului unei zone geografice se pot folosi coordonate geocentrice, coordonate geodezice sau coordonate într-un sistem de proiecție cartografic cunoscut (UTM – *Universal Transversal Mercator*, *Lambert*, etc).

În general, toate aceste sisteme de coordonate folosesc datele de referință ale globului pământesc standard WGS84. Sistemul de coordonate geocentric este un sistem de coordonate tridimensional cu originea în centrul Pământului. Coordonatele geodezice sunt coordonatele date în latitudine și longitudine. Coordonatele UTM se obțin prin proiecția suprafeței Pământului pe un cilindru care înfășoară globul pământesc de-a lungul unui anumit meridian. Cilindrul Mercator este desfășurat într-un plan în care este definit un sistem de referință bidimensional ( $x, y$ ), cu axa  $x$  către Sud și axa  $y$  către Est, la care se adaugă axa  $z$  de coordonate pentru altitudine, pentru obținerea unui sistem de coordonate tridimensional drept. În coordonate UTM, eroarea de proiecție crește cu cât crește distanța față de meridianul de definiție al proiecției.

Sistemul de coordonate terestre (geocentric, geodezic sau UTM) este considerat sistem de referință universal în reprezentarea scenelor virtuale. În unele sisteme de realitate virtuală se definesc în mod diferit axele sistemului de referință universal față de modul definit mai sus. De exemplu, sistemul de referință universal poate fi definit cu axa  $x$  este îndreptată către Est, axa  $z$  către Sud, iar axa  $y$  în sus (spre cer). Un alt mod de interpretare care poate fi întâlnit în unele sisteme grafice, biblioteci sau programe de modelare este cu axa  $x$  către Nord, axa  $y$  către Est și axa  $z$  îndreptată în jos (către pământ). Nu se poate spune că o convenție este mai bună decât alta, dar trebuie cunoscută convenția folosită și efectuate transformările necesare între diferitele sisteme de referință care intervin în sistemul proiectat. În expunerea care urmează, se consideră un sistem de coordonate cu axa  $x$  către Sud și axa  $y$  către Est, iar axa  $z$  reprezintă altitudinea ( $z$ ).

Datele de teren pot fi obținute din hărți digitale (*Digital Terrain Elevation Data* – DTED) care memorează altitudinea într-o grilă spațială regulată, într-un sistem de coordonate geodezice. Distanța dintre puncte este dată în arc-secundă pe latitudine și longitudine, cu valori care diferă în funcție de rezoluția datelor (1 arc-secundă sau 3 arc-secundă). La trecerea din coordonate geodetice în coordonate UTM, distanța între puncte nu se mai păstrează constantă, deci grila de puncte nu mai este regulată. Deoarece datele de aproximare a suprafeței terenului sunt altitudini cunoscute într-o grilă de puncte  $p(x_i, y_i)$ , orice linie paralelă cu axa  $z$  intersectează suprafața terenului într-un singur punct și suprafața terenului poate fi construită pornind de la o funcție două variabile  $z = F(x, y)$ , definită într-un domeniu discret  $\psi = \{p(x_i, y_i)\}$  [Cig97].

Construirea rețelei poligonale de aproximare a suprafeței astfel definite se poate face prin triangularizare în planul bidimensional, după care se adaugă altitudinea  $z_i$  a fiecărui punct. În acest mod se obține o suprafață poligonală



deschisă, care nu respectă condiția de închidere a suprafețelor de frontieră a poliedrelor, dar această condiție nu este necesară în reprezentarea terenului.

În fig. 2.13 este reprezentată suprafața unei zone geografice de teren, obținută prin triangularizarea tuturor punctelor grilei de altitudini din harta digitală corespunzătoare acelei zone.

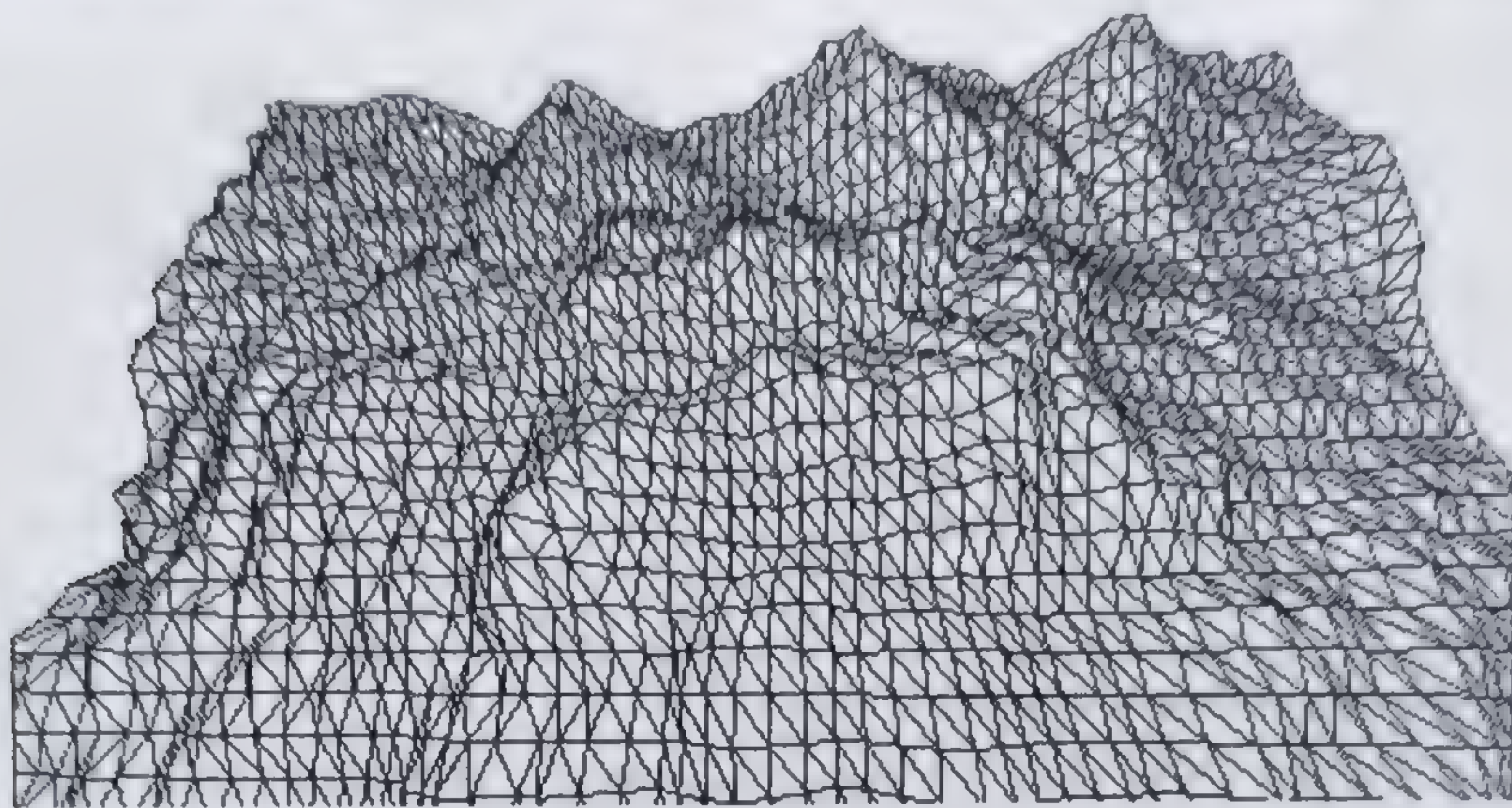


Fig. 2.13 Triangularizarea unei suprafețe de teren.

Triangularizarea prin considerarea tuturor punctelor din domeniul dat creează o funcție de aproximare  $F^*(x, y)$  liniară pe porțiuni care aproximează suprafața dată prin funcția  $F(x, y)$  cu eroare 0 în tot domeniul de definiție. O astfel de soluție este în general nesatisfăcătoare, datorită volumului mare de date rezultat. Terenul din fig. 2.14 a fost definit pe o grilă de  $61 \times 61$  puncte cu rezoluția de 3 arc-secunde și au rezultat 7200 de triunghiuri, dat fiind că s-au inserat toate punctele în triangularizare. Costul de stocare și prelucrare a acestor date poate fi inacceptabil, iar, pe de altă parte, se poate ca variațiile locale mici să nu prezinte interes în aplicațiile de realitate virtuală.

Alternativa o constituie construirea unei triangularizări care utilizează numai o parte din punctele domeniului de definiție și care respectă condiția de precizie impusă, formulată astfel:

$$|F^*(x, y) - F(x, y)| \leq \delta, \forall p(x, y) \in \psi \quad (2.7)$$

unde  $\delta$  reprezintă eroarea maximă impusă (fig. 2.14).

Una din metodele de construire a unui model care să respecte condiția 2.7 este metoda selecției punctelor [Fjall93], care calculează o secvență  $T_0, T_0, \dots, T_m$  de triangularizări pentru submulțimi succesiv crescătoare ale mulțimii  $\psi$  a datelor de intrare. În fig. 2.14 s-au reprezentat proiecțiile în planul  $xy$  ale unor triangularizări prin metoda selecției punctelor. În triangularizarea inițială  $T_0$  se consideră punctele de pe frontiera domeniului și se poate folosi orice metodă de triangularizare Delaunay în plan cunoscută (metoda cercului vid, metoda incrementală, etc.). După triangularizarea plană se adaugă coordonata  $z$  fiecărui punct din diagrama de triangularizare.



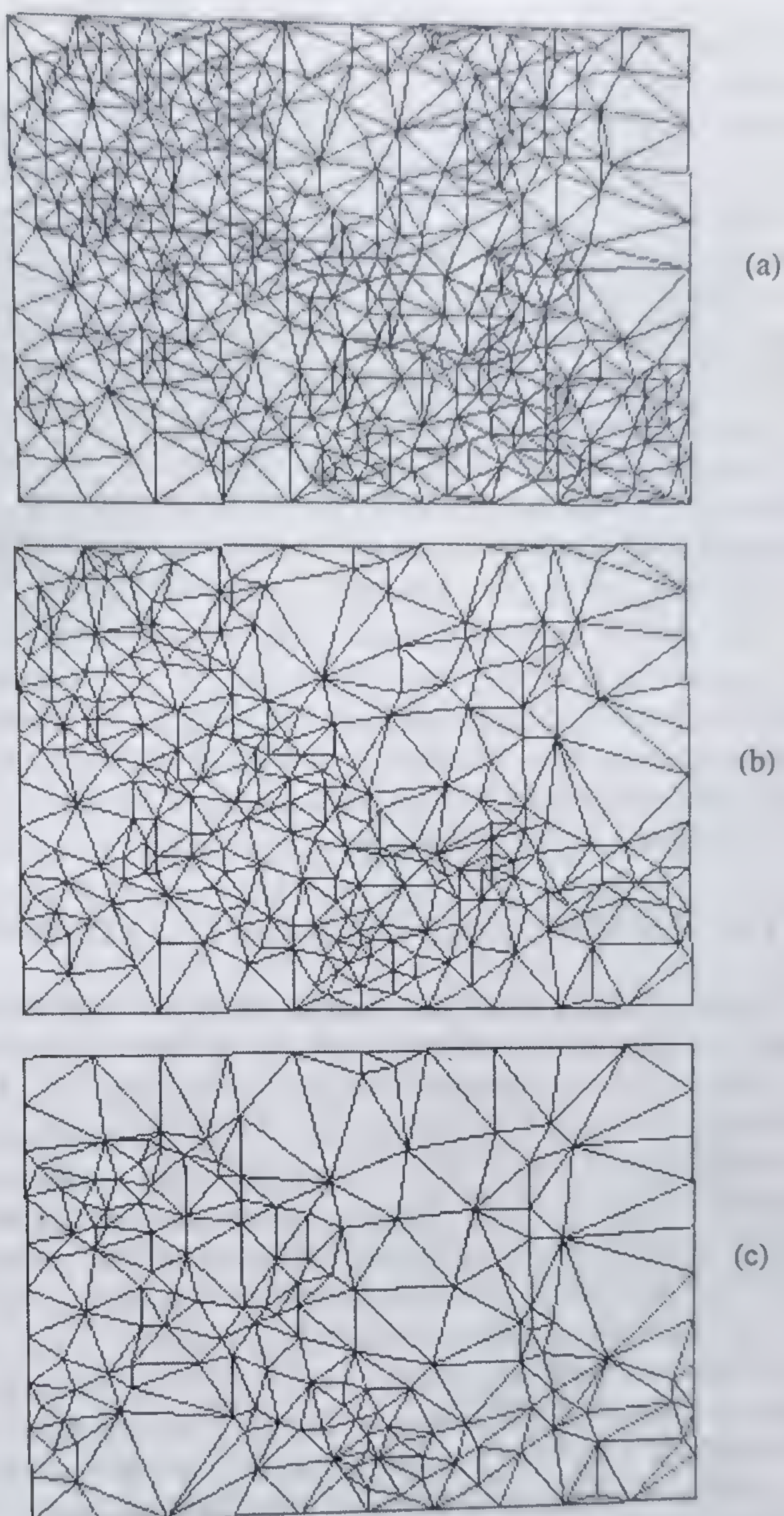


Fig. 2.14 Triangularizarea datelor de teren cu eroare maximă impusă.  
(a)  $\delta = 50$ ; (b)  $\delta = 70$ ; (c)  $\delta = 90$ .

Fiecărei triangularizări  $T_k$  îi corespunde o funcție liniară pe porțiuni  $F_k$ , astfel că, în interiorul fiecărui triunghi al triangularizării  $T_k$ ,  $F_k$  este dată de ecuația planului ce trece prin vârfurile triunghiului respectiv. Pentru fiecare triangularizare  $T_k$  există o locație  $p_i(x_i, y_i)$  pentru care distanța la planul corespunzător funcției  $F_k$  (numită deviație) este maximă. Dacă în triangularizarea  $T_k$  deviația maximă este



mai mare decât eroarea maximă impusă  $\delta$ , atunci se construiește o nouă triangularizare  $T_{k+1}$  prin adăugarea în diagrama de triangularizare a punctului  $p_i$ . Dacă deviația maximă este mai mică sau egală cu eroarea maximă impusă, triangularizarea  $T_k$  este triangularizarea finală, iar funcția  $F_k = F^*(x, y)$  pentru eroarea impusă  $\delta$ .

În fig. 2.14 s-au reprezentat proiecțiile în planul  $xy$  ale triangularizărilor având erori maxime impuse de 50, 70, 90, respectiv. Față de triangularizarea inițială care a produs 7200 de triunghiuri, triangularizările (a), (b), (c) au 940, 512, 300 triunghiuri, ceea ce constituie reprezentări eficiente în condițiile de eroare impuse. Controlul erorii de aproximare a suprafeței obiectelor permite construirea modelelor cu nivele multiple de detaliu (*levels of detail* – LOD).

Modelul este constituit dintr-o colecție de  $m$  nivele de detaliu, de la nivelul 0 (detaliere maximă), până la nivelul  $m - 1$  (detaliere minimă). Un nivel de detaliu se definește prin eroarea de aproximare maximă impusă  $\delta$ . În cursul redării imaginii obiectului, se selectează nivelul de detaliu corespunzător, în funcție de distanța obiectului față de punctul de observare. Cu cât obiectul este mai depărtat de punctul de observare, cu atât imaginea acestuia este de dimensiune mai mică în proiecția perspectivă, și se poate selecta un nivel de detaliu cu mai puține elemente.

Cele trei reprezentări ale terenului din fig. 2.14 pot fi folosite ca nivele de detaliu în modelarea acestei zone geografice.

### 2.1.8 REDAREA IMAGINII OBIECTELOR POLIGONALE

Așa cum s-a mai amintit, modelarea poligonală este cea mai folosită formă de modelare a obiectelor în grafica pe calculator, datorită simplității reprezentării modelului poligonal și faptului că, în momentul de față, majoritatea acceleratoarelor din sistemele grafice conțin programe implementate hardware de redare eficientă a poligoanelor. Algoritmii de redare implementați în acceleratoarele grafice tratează separat fiecare poligon (primitivă geometrică) reprezentat prin coordonatele vârfurilor sale, ceea ce face ca procesul de redare a imaginii să fie foarte simplu. Redarea imaginilor scenelor virtuale compuse din obiecte a căror reprezentare însumează mii de poligoane nu este ceva neobișnuit.

Redarea obiectelor poligonale este implementată printr-o succesiune de operații de transformări grafice asupra obiectelor numită *pipeline grafic*. Această succesiune, care va fi descrisă pe larg în capitolul următor, constă din transformări geometrice aplicate vârfurilor obiectelor, prin care se transformă fiecare față a obiectului din sistemul de referință de modelare într-un sistem de referință de afișare, urmată de transformarea de redare, prin care se obține culoare pixelilor care se afișează pe display. În fig. 2.15 este redată imaginea avionului F-16 modelat prin 2413 fețe poligonale și redat prin prelucrarea fiecărei fețe, în modul cu fețe "pline" și umbrire, și în modul "wireframe". Pentru toate celelalte modele de reprezentare a obiectelor (reprezentarea prin rețele de petice, prin compunerea obiectelor sau prin divizarea spațială), pe lângă redarea directă a modelului respectiv, există și posibilitatea de redare prin deducerea mai întâi a reprezentării poligonale corespunzătoare, urmată de folosirea algoritmilor de redare poligonală.



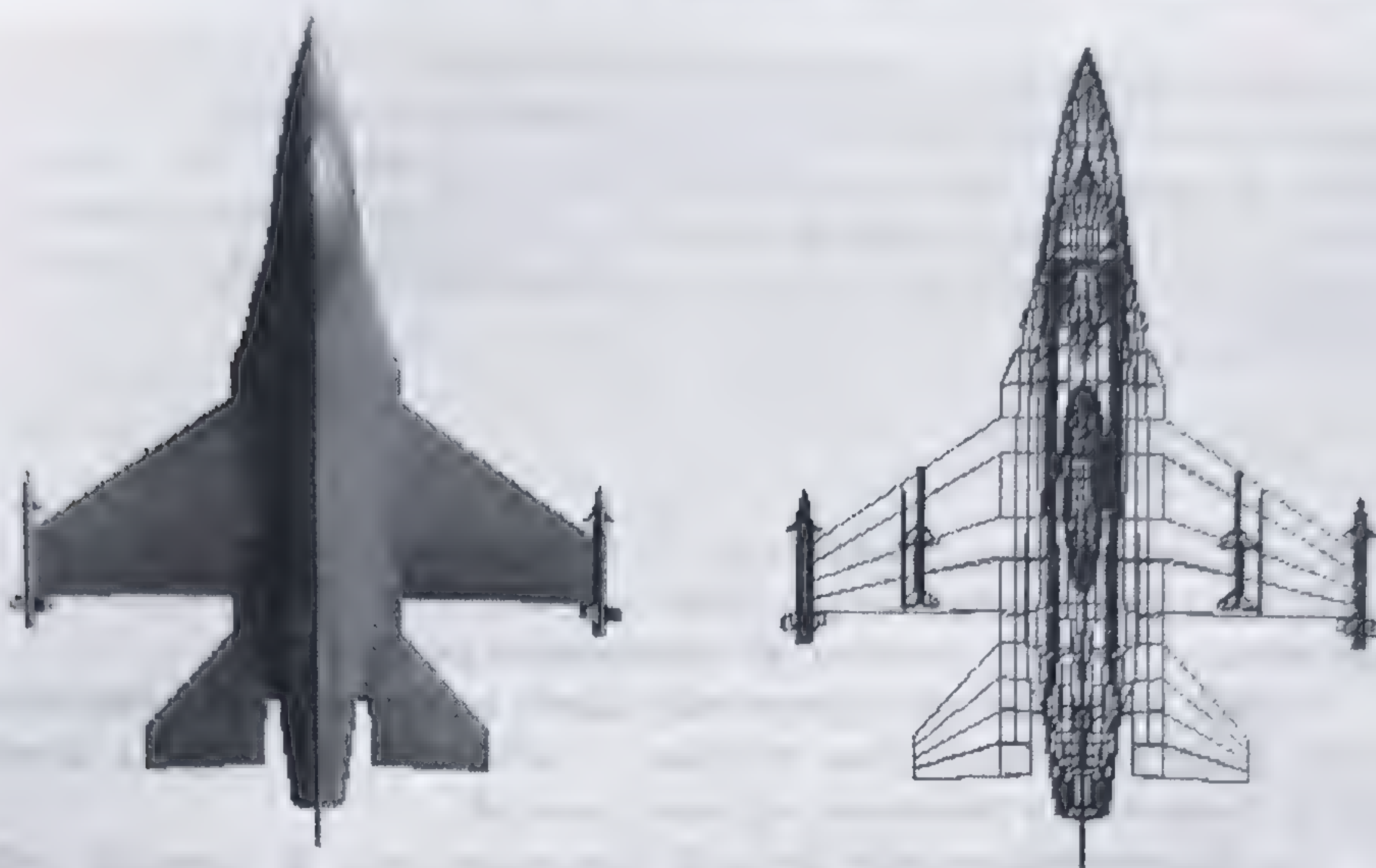


Fig. 2.15 Redarea obiectelor modelate prin rețea de fețe poligonale.

## 2.2 MODELAREA OBIECTELOR PRIN SUPRAFETE PARAMETRICE

Un petic (*patch*) este o suprafață curbă definită parametric în spațiul tridimensional. Prin reprezentarea parametrică, punctele de pe suprafață se pot calcula secvențial, pentru diferite valori ale parametrilor, mult mai simplu decât prin rezolvarea sistemului de ecuații care descriu implicit suprafața. Reprezentarea parametrică a curbilor și a suprafețelor este folosită în proiectarea și în modelarea obiectelor, pentru obținerea unei precizii mai ridicate de aproximare.

Un segment de curbă în spațiul tridimensional poate fi definit printr-un sistem de ecuații în funcție de un parametru:

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \quad (2.8)$$

Aceasta este o curbă cubică, definită ca formă și mărime de cei 12 coeficienți, numiți coeficienți algebrici ai curbei. În notație vectorială se poate scrie forma parametrică a unei curbe cubice:

$$Q(u) = Au^3 + Bu^2 + Cu + D \quad (2.9)$$

unde vectorul  $Q$  are componentele  $(x, y, z)$ , iar vectorii  $A, B, C, D$  au componentele  $(a_x, a_y, a_z), (b_x, b_y, b_z), (c_x, c_y, c_z), (d_x, d_y, d_z)$ , respectiv, iar  $u$  este cuprins în intervalul închis  $[0, 1]$ .



În aplicațiile grafice se utilizează în mod frecvent curbele cubice, deoarece ele asigură suficientă flexibilitate în definirea curbelor și pot fi prelucrate eficient. Polinoame de grad mai mare pot descrie curbe mai complexe, dar necesită un număr mai mare de coeficienți și sunt mai dificil de prelucrat eficient în reprezentările grafice. La modul general, o curbă parametrică cubică se poate defini prin ecuațiile:

$$\mathbf{Q}(u) = \sum_{i=0}^3 \mathbf{P}_i \mathbf{B}_i(u) \quad (2.10)$$

unde  $\mathbf{Q}$  este vectorul de componente  $(x, y, z)$ ,  $\mathbf{P}_i$  sunt punctele de control ale curbei, fiecare vector  $\mathbf{P}_i$  având componentele  $(x_i, y_i, z_i)$ , iar  $\mathbf{B}_i$  sunt funcțiile de bază (sau de amestec – *blending functions*) ale reprezentării parametrice.

Diferite funcții de bază au proprietăți diferite în determinarea formei curbei parametrice: ele pot interpola sau aproxima o curbă dată, pot asigura anumite condiții de continuitate a mai multor segmente de curbă.

Specificarea unui segment de curbă (sau a unui petic de suprafață curbă) printr-un set de puncte de control este o metodă de bază în proiectarea grafică interactivă: proiectantul definește punctele de control; curba este generată și vizualizată interactiv; dacă forma curbei nu este mulțumitoare, proiectantul modifică unul sau mai multe puncte de control, până obține rezultatul dorit.

Cele mai utilizate tipuri de curbe și suprafețe parametrice în proiectarea grafică sunt curbele (și suprafețele) Bézier și B-spline. Modelarea curbelor și a suprafețelor parametrice este prezentată detaliat în capitolul 8.

## 2.3 MODELAREA PRIN COMPUNEREA OBIECTELOR

Modelarea prin compunerea obiectelor (*Constructive Solid Geometry* – CSG) se folosește atunci când un obiect poate fi obținut prin combinarea mai multor obiecte elementare, numite primitive geometrice.

Primitivele geometrice utilizate sunt sfere, conuri, cilindri sau paralelipipede dreptunghice și sunt combinate folosind operatori booleani și transformări liniare. Un obiect complex este reprezentat printr-un arbore, ale cărui frunze sunt primitivele geometrice, iar nodurile memorează operatori booleani sau transformări liniare. În fig. 2.16 este prezentată operația de reuniune a două obiecte elementare.

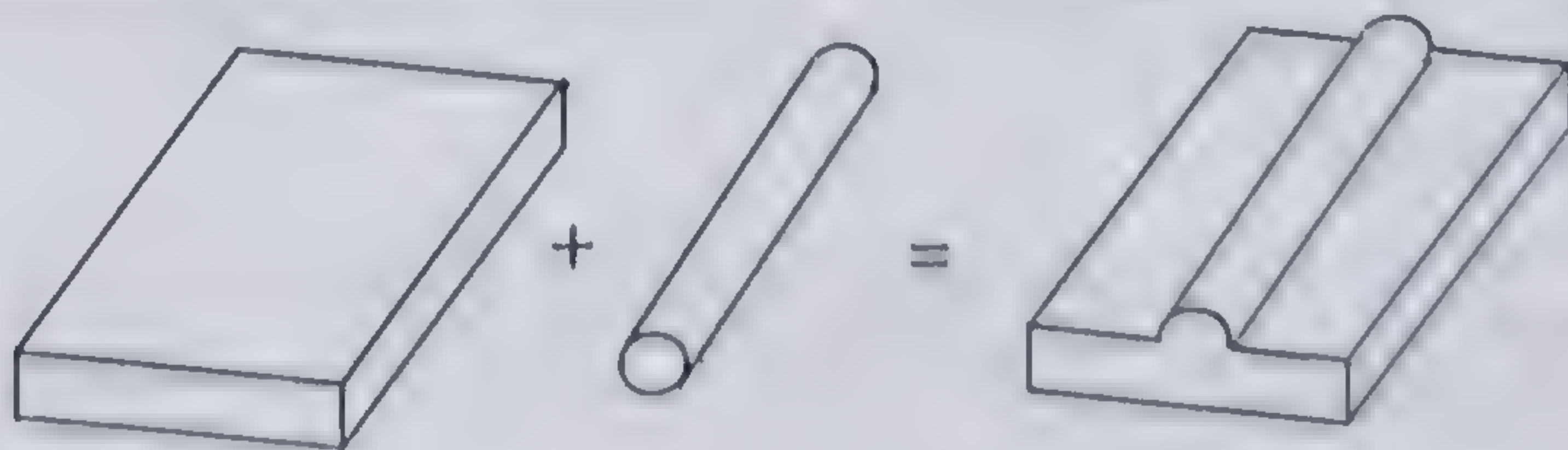


Fig. 2.16 Operația de reuniune a două primitive geometrice.



Alte operații posibile care se pot efectua asupra primitivelor geometrice sunt scăderea și intersecția. Fig. 2.17 arată reprezentarea CSG a unui obiect prin combinarea a trei obiecte elementare: două paralelipipede dreptunghice și un cilindru. Arborele de reprezentare are ca frunze primitivele geometrice, iar celelalte noduri conțin operatorii booleni. Paralelipipelele sunt combinate folosind operația de reuniune, iar o gaură este practică într-unul din paralelipipele prin scăderea unui cilindru din ansamblul celor două paralelipipele.

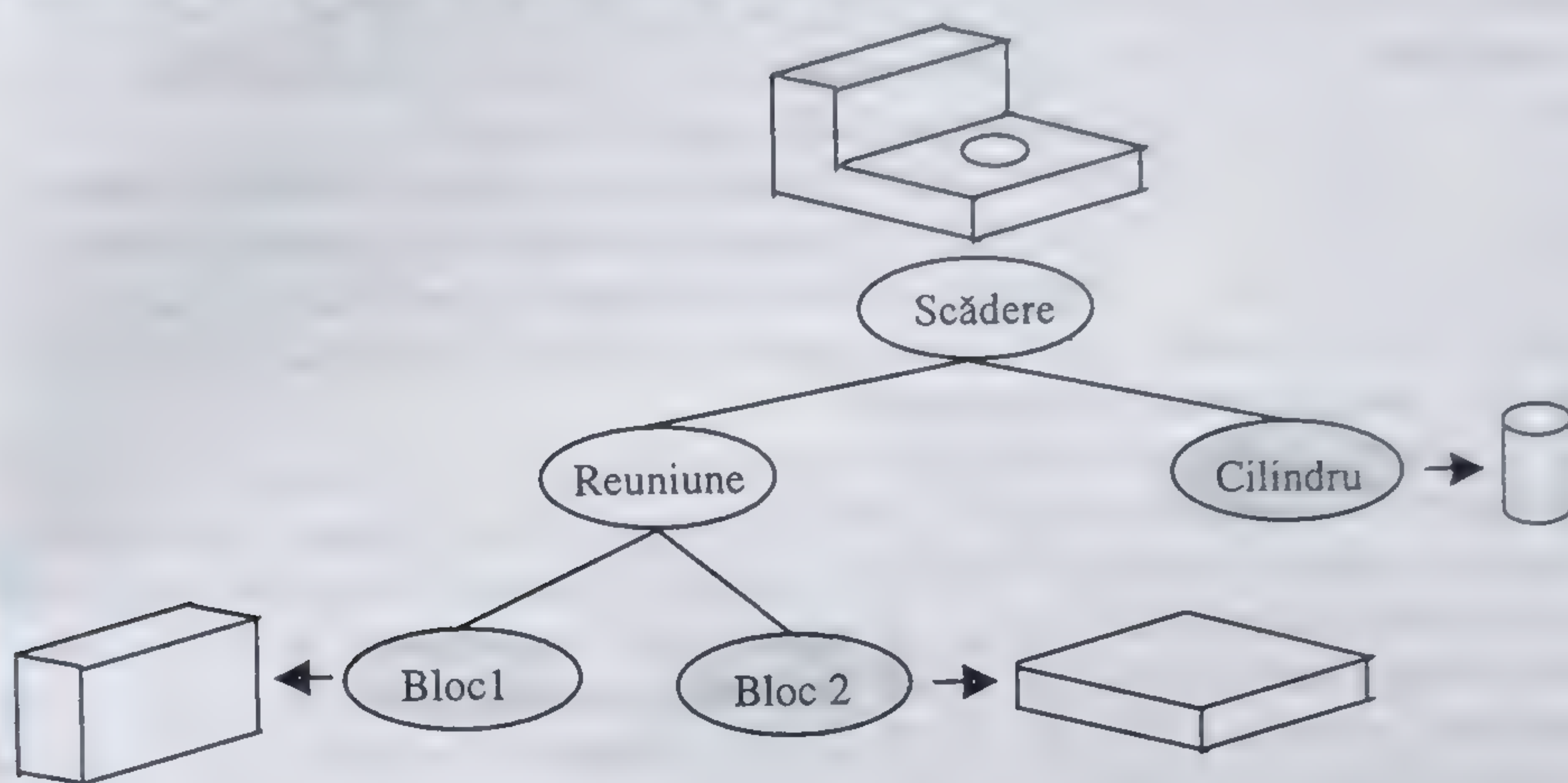


Fig. 2.17 Arborele de construire a unui obiect din trei primitive geometrice.

Deoarece arborele de reprezentare memorează atât operațiile boolene, cât și forma primitivelor geometrice, operațiile de modificare ale obiectului compus sunt relativ simple. De exemplu, o gaură într-un obiect se modifică prin modificarea poziției sau a dimensiunii primitivei geometrice folosite în operația de scădere. O astfel de modificare este mult mai dificilă în reprezentarea prin suprafața de frontieră a modelelor.

**Redarea imaginii obiectelor CSG** se poate face prin mai multe metode:

- Redarea directă a obiectului în aplicațiile de *ray-tracing*.
- Conversia modelului CSG în reprezentare prin suprafața de frontieră (B-rep) și aplicarea procedurilor standard de redare a poligoanelor.
- Conversia modelului CSG în reprezentare prin divizarea spațiului (în elemente numite voxel) și redarea volumului corespunzător

Tehnica *ray-tracing* generează imaginea obiectelor prin intersecția acestora cu raze de proiecție și permite obținerea unor efecte deosebit de realiste în modelarea reflexiei, transparenței și a umbrelor.

Un dezavantaj important al modelării CSG este acela că nu orice fel de obiect poate fi obținut prin combinarea unor primitive geometrice simple. De exemplu, modelul unei statui (folosit în crearea muzeelor virtuale) sau modelul unui organ anatomic (folosit în simulările de intervenții chirurgicale) nu poate fi obținut prin combinarea unor primitive geometrice simple.



## 2.4 MODELAREA PRIN DIVIZARE SPAȚIALĂ

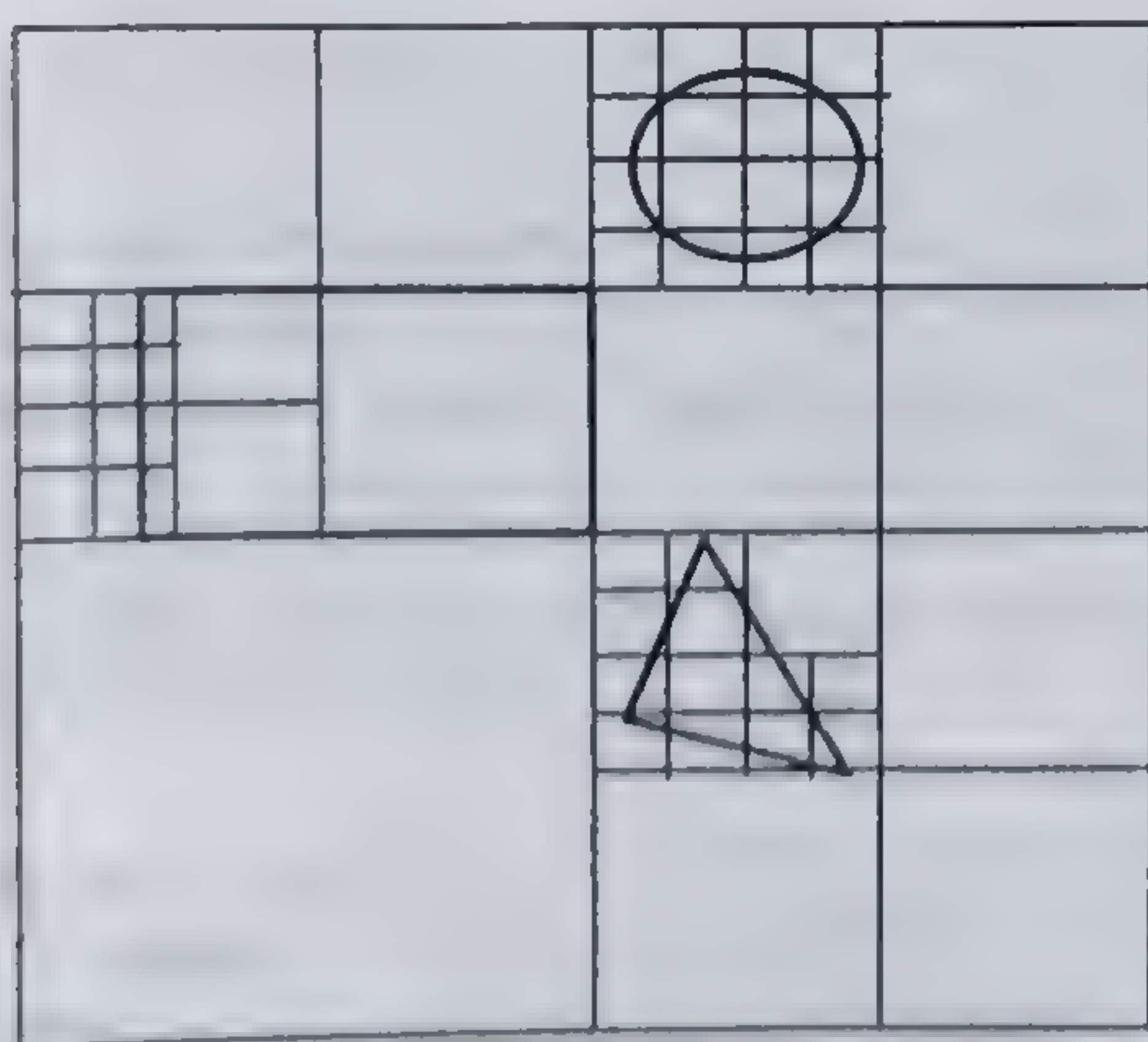
În tehnica de divizare spațială, se atribuie fiecărei subdiviziuni a spațiului tridimensional câte o etichetă în funcție de obiectul care ocupă acea subdiviziune.

În această tehnică se consideră spațiul tridimensional compus dintr-un număr de  $m \times n \times k$  volume elementare (numite *voxeli*) și pentru fiecare voxel se memorează eticheta (un număr de identificare) obiectului care ocupă acel element. Reprezentarea prin subdiviziunea spațiului este utilă în diferite aplicații grafice, cum sunt imagistica medicală sau aplicațiile de *ray tracing*.

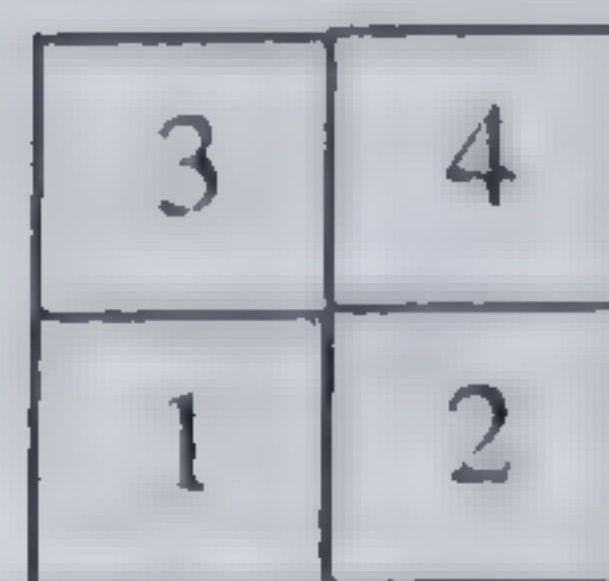
Implementarea directă, prin atribuirea unei etichete fiecărui element de volum al spațiului, implică o mare redundanță a datelor: toți voxelii interiori unui obiect conțin aceeași informație, eticheta obiectului. Acest lucru conduce la o cantitate de informație foarte mare necesară pentru reprezentarea unui spațiu de obiecte și este, în general, inacceptabilă.

Pentru reducerea redundanței de reprezentare se folosește o tehnică de reprezentare a ocupării spațiale printr-un arbore numit arbore octal (*octree*). Arborele octal este o structură ierarhică care specifică ocuparea unei regiuni cubice din spațiul tridimensional. Ideea de reprezentare a ocupării spațiale poate fi exemplificată mai ușor într-un spațiu bidimensional prin intermediul unui arbore cuaternar (*quadtree*). În fig. 2.18(a) este prezentată ocuparea unei regiuni bidimensionale de trei obiecte, iar în fig. 2.19 este reprezentat arborele de ocupare a regiunii.

Arborele se creează pornind cu o regiune pătrată în plan, constituind întreaga zonă care se modelează și care este reprezentată prin nodul rădăcină al arborelui cuaternar. În cazul spațiului tridimensional, această regiune este un cub. Fiecare regiune, începând cu regiunea inițială, se subdivide în patru subregiuni, reprezentate ca patru noduri fii în arbore. În fig. 2.18(b) se arată ordinea de numerotare a nodurilor fii obținuți prin divizarea unei subregiuni.



(a)



(b)



Fig. 2.18 (a) Ocuparea unei regiuni plane.  
(b) Ordinea de numerotare a nodurilor fii.



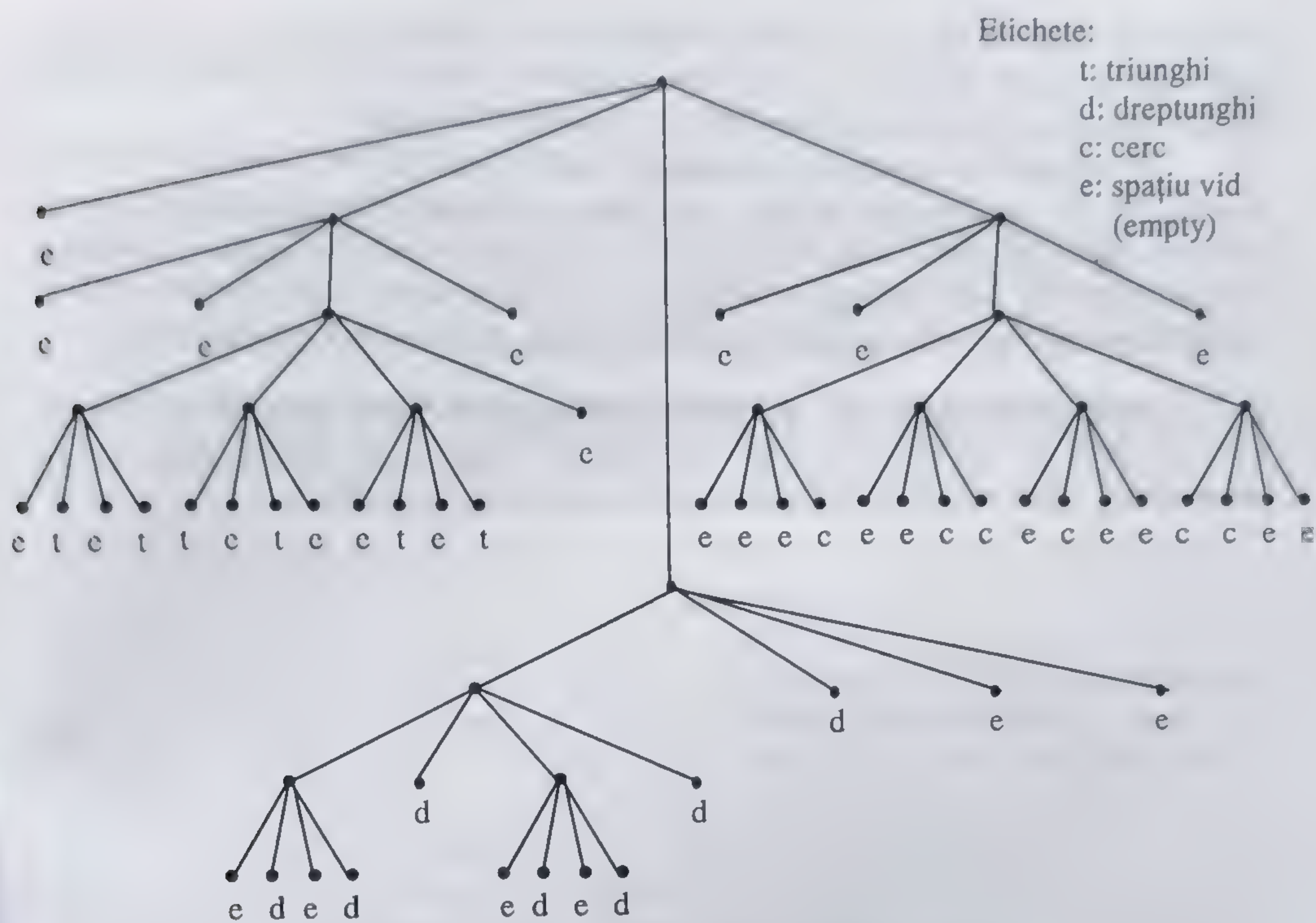


Fig. 2.19 Arborele de reprezentare a ocupării spațiale.

În spațiul tridimensional o regiune este divizată în opt subregiuni, reprezentate prin opt noduri fii; de aici provine și numele (*arbore octal*) acestui mod de reprezentare.

Subregiunile sunt divizate recursiv până se întâlnește una din următoarele situații:

- Subregiunea este ocupată în întregime de un singur obiect sau nu este ocupată de nici un obiect. Unei astfel de regiuni i se atribuie eticheta cu numele obiectului respectiv, sau eticheta de spațiu vid (e).
- Subregiunea are dimensiunea minimă admisibilă în reprezentarea respectivă (un pixel). În acest caz, subregiunea primește eticheta obiectului (sau a spațiului vid) care ocupă cea mai mare parte din subregiunea respectivă.

În arborele de reprezentare a ocupării spațiului tridimensional, dimensiunea minimă până la care se divizează subregiunile este dimensiunea unui voxel. În arborele de reprezentare sunt două tipuri de noduri frunză: noduri care au o etichetă a unui obiect sau noduri care au eticheta spațiului vid. Compactarea volumului de date în această reprezentare față de reprezentarea prin etichetarea fiecărui element (pixel sau voxel) provine din faptul că se poate ca o întreagă subregiune (mai mare decât un voxel) să primească o singură etichetă, dacă este



ori nu este ocupată deloc, sau este ocupată de un singur obiect. În fig 2.18 (a), regiunea din partea stânga jos din prima diviziune nu mai este subdivizată mai departe deoarece este în întregime vidă și primește eticheta (e).

O variantă de reprezentare a ocupării spațiale prin arbori octali folosește ca element spațial un tetraedru regulat, care este subdivizat în opt tetraedre, până la rezoluția dorită sau până când este ocupat de un singur obiect din spațiu. Avantajul folosirii arborilor octali bazați pe tetraedre este acela că se poate obține direct o reprezentare echivalentă a suprafeței triangularizate de aproximare a obiectului.

**Redarea imaginii obiectelor modelate prin divizare spațială** se face fie prin conversie în reprezentarea prin suprafața de frontieră, fie direct, prin *redare volumetrică*, adică redarea tuturor elementelor de volum (voxeli).



## TRANSFORMĂRI GEOMETRICE

În procesul de redare a obiectelor tridimensionale se definesc mai multe sisteme de referință, care permit descrierea operațiilor de transformări succesive ale obiectelor, pornind de la modelul acestora, până la imaginea bidimensională pe suportul de reprezentare.

Aceste sisteme de referință vor fi precizate pe parcursul expunerii; ceea ce interesează în acest moment este faptul că în vizualizarea scenelor virtuale sunt necesare (și folosite) mai multe sisteme de referință în care sunt reprezentate obiectele. Modificarea unui obiect într-un sistem de referință dat, sau trecerea de la un sistem de referință la altul, se realizează prin diferite transformări grafice. Dintre transformările grafice folosite, unele modifică forma și poziția obiectelor în spațiu, fiind numite transformări geometrice, altele sunt transformări de conversie între diferite modalități de reprezentare a obiectelor (transformarea de rastru).

### 3.1 TRANSFORMĂRI GEOMETRICE ÎN SPAȚIU

Obiectele scenei virtuale pot fi modificate sau manevrate în spațiul tridimensional folosind diferite transformări geometrice. Dintre acestea, cele mai importante sunt: *translația*, care modifică localizarea obiectului; *rotația*, care modifică orientarea; *scalarea*, care modifică dimensiunea obiectului. Aceste transformări sunt denumite transformări geometrice primitive.

#### 3.1.1 TRANSFORMĂRI GEOMETRICE PRIMITIVE

**Translația** este transformarea prin care toate punctele se deplasează în aceeași direcție și cu aceeași distanță între punct și transformatul său. Translația se poate descrie printr-un vector de translație  $T$ , având componentele  $t_x$ ,  $t_y$ ,  $t_z$  pe cele trei axe de coordonate; un punct  $P(x,y,z)$  se transformă în punctul  $P'(x',y',z')$  astfel:



$$\begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases}$$

În notație matriceală, transformarea prin translație cu vectorul de translație  $T$ , având componentele  $t_x, t_y, t_z$  pe cele trei axe de coordonate a unui punct  $P(x, y, z)$  în punctul  $P'(x', y', z')$  se exprimă printr-o însumare de matrice:

$$P' = P + T, \text{ unde } T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}; \text{ deci: } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

**Scalarea** modifică coordonatele tuturor punctelor unui obiect folosind factorii de scală  $s_x, s_y$ , respectiv  $s_z$  pe cele trei axe de coordonate. În această transformare de scalare, numită scalare față de origine, un punct  $P(x, y, z)$  se transformă în punctul  $P'(x', y', z')$ , unde:

$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \\ z' = z \cdot s_z \end{cases}$$

Pentru scrierea sub formă matriceală a acestor relații de transformare, se definește matricea de scalare  $S$  de dimensiune  $3 \times 3$  astfel:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Rezultă relația de transformare de scalare în notație matriceală:

$$P' = SP, \text{ adică } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

**Rotația** cu un unghi  $\theta$  în raport cu o axă  $D$  este o transformare prin care orice punct  $P$  care nu este situat pe dreapta  $D$  se transformă într-un punct  $P'$  astfel încât  $P$  și  $P'$  sunt situate într-un plan perpendicular  $\pi$  pe dreapta  $D$ , unghiul  $POP'$  este egal cu  $\theta$ , iar modulele  $|OP|$  și  $|OP'|$  sunt egale (punctul  $O$  este intersecția dintre dreapta  $D$  și planul  $\pi$ ). Prin această transformare toate punctele dreptei  $D$  sunt fixe și numai ele sunt puncte invariante ale transformării. Transformarea de rotație în raport cu o axă oarecare se descompune într-o succesiune de maximum trei transformări de rotație în raport cu axele de coordonate ale sistemului de referință.

Rotația în raport cu axa  $z$  cu un unghi  $\theta$  transformă un punct  $P(x, y, z)$  în punctul  $P'(x', y', z')$ , ambele aflate în planul  $\pi$  perpendicular pe axa  $z$ . Pentru



deducerea relațiilor de transformare se exprimă coordonatele punctelor  $P$  și  $P'$  în acest plan în coordonate polare (fig. 3.1).

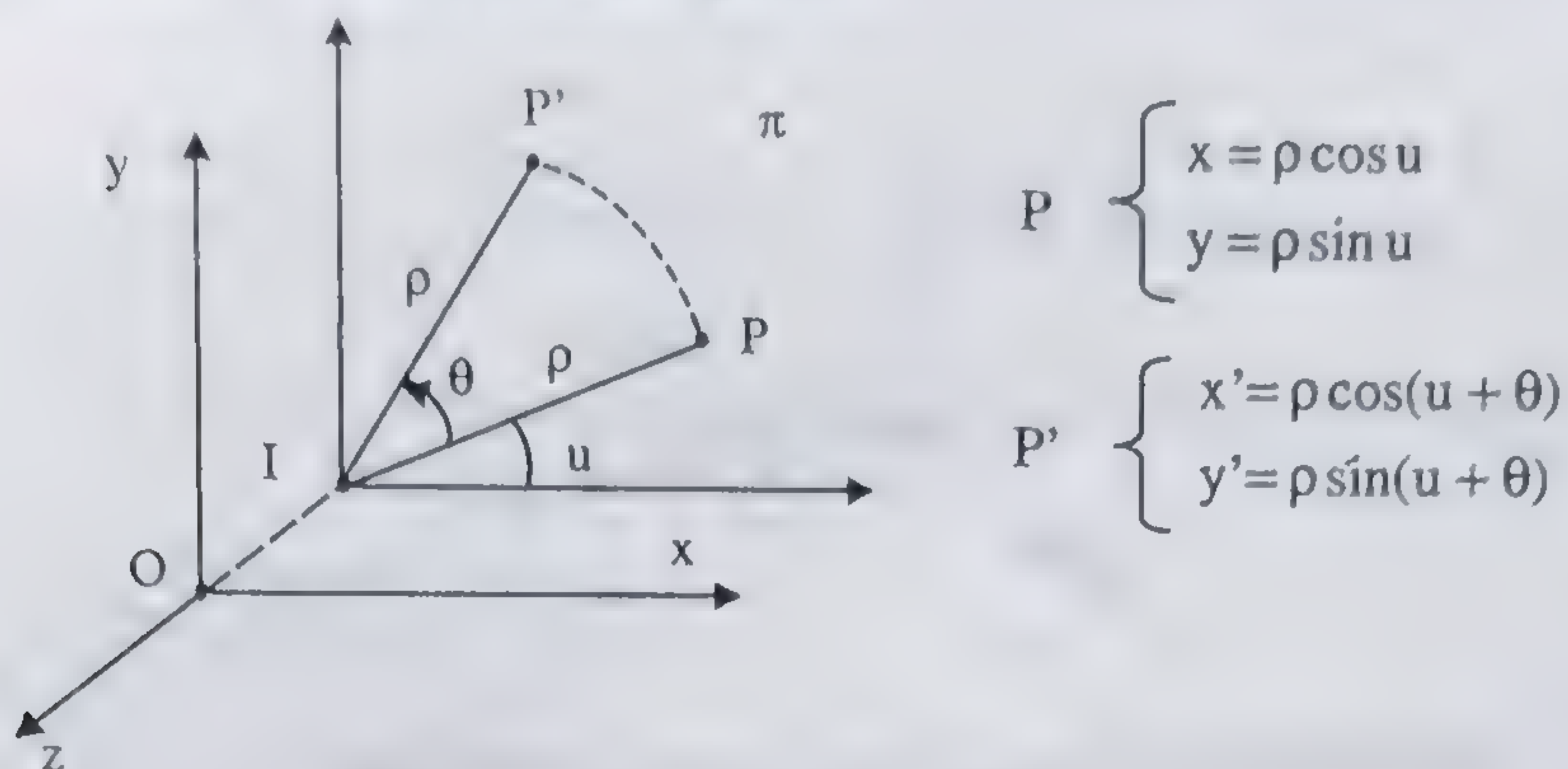


Fig. 3.1 Transformarea de rotație în raport cu axa  $z$ .

Se dezvoltă cosinusul și sinusul sumei de unghiuri și se obțin relațiile:

$$x' = \rho(\cos u \cos \theta - \sin u \sin \theta) = x \cos \theta - y \sin \theta$$

$$y' = \rho(\sin u \cos \theta + \sin \theta \cos u) = x \sin \theta + y \cos \theta$$

Această transformare se poate scrie sub formă matriceală dacă se definește matricea de rotație  $R_z(\theta)$  de dimensiune  $3 \times 3$  astfel:

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rezultă relațiile de transformare de rotație a unui punct în raport cu axa  $z$  cu un unghi  $\theta$  scrise sub formă matriceală:

$$P' = R_z P, \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Pentru rotațiile elementare ale unui punct în raport cu axele  $x$  și  $y$  ale sistemului de referință se urmărește un raționament asemănător și se deduc relațiile de transformare corespunzătoare.

Rotațiile în raport cu axele de coordonate ale sistemului de referință sunt denumite tangaj (*pitch*), girație (*yaw, heading*) și, respectiv, ruliu (*roll*). Această asignare depinde de convențiile de definire a sistemului de referință universal. Pentru convenția definită mai sus, tangajul este o rotație în raport cu axa  $x$ , girația este o rotație în raport cu axa  $y$ , iar ruliul este o rotație în raport cu axa  $z$  (fig. 3.2).

Aceste denumiri au originea în modul în care sunt definite mișcările unui avion poziționat în spațiu cu axa longitudinală orientată către  $z$  pozitiv: tangajul este rotația într-un plan vertical, care înclină botul avionului; girația este o mișcare



într-un plan orizontal, care schimbă direcția axei avionului, iar ruliul este rotația într-un plan vertical, care înclină aripile avionului.

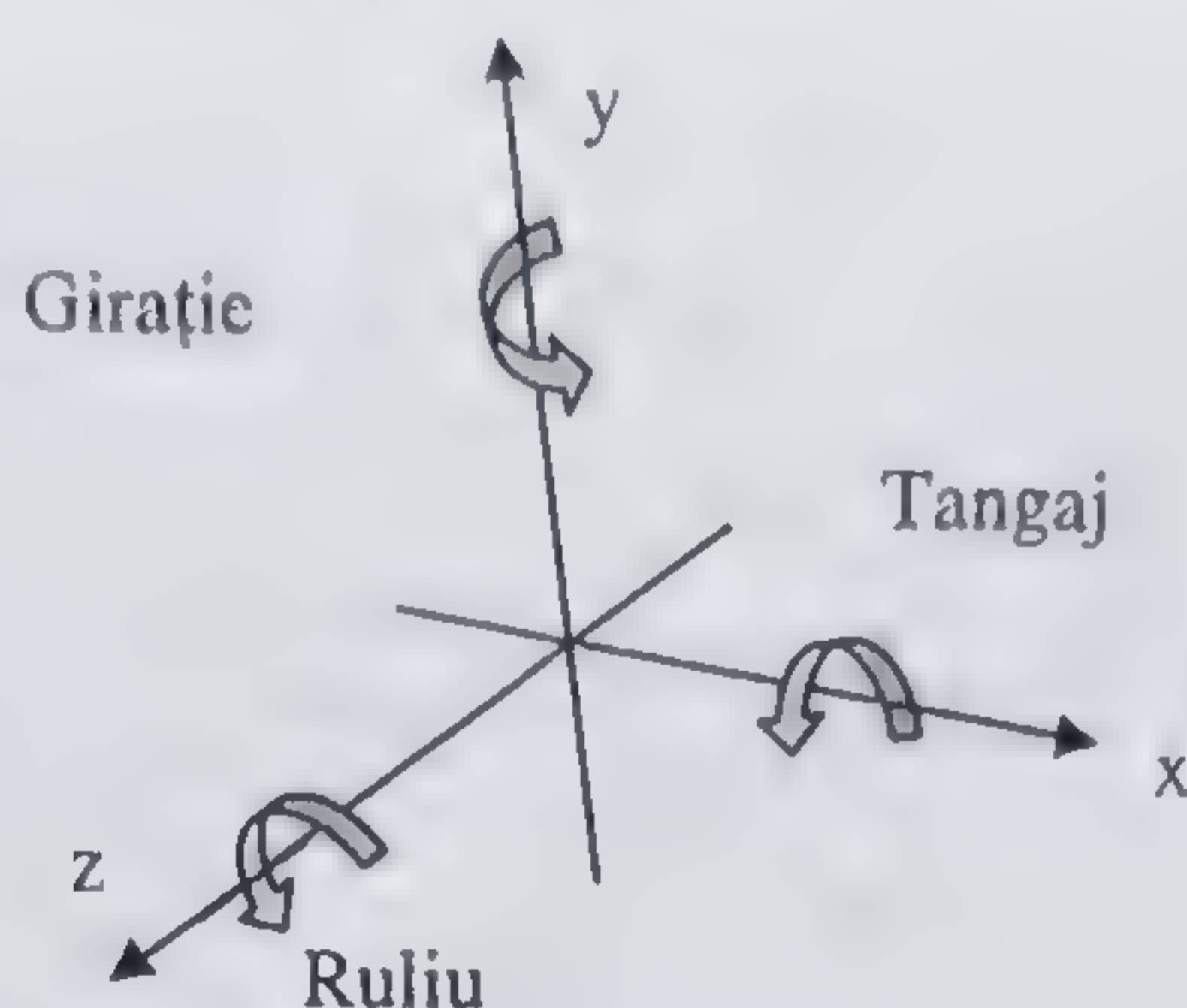


Fig. 3.2 Rotațiile în raport cu axele sistemului de referință.

Semnul rotațiilor în raport cu axele de coordonate se alege, prin convenție, după regula mâinii drepte sau după regula mâinii stângi. Dacă se cuprinde axa corespunzătoare cu patru degete ale mâinii (drepte, respectiv, stângi) astfel ca degetul mare să fie îndreptat în sensul pozitiv al axei, direcția celor patru degete indică sensul pozitiv al rotației. În această lucrare se adoptă regula mâinii drepte pentru sensul de rotație în raport cu axele de coordonate (fig. 3.2).

S-au obținut relațiile pentru transformările spațiale elementare (primitive), translația, scalarea față de originea sistemului de coordonate și rotația în raport cu axele sistemului de coordonate. Tratarea într-un mod unitar a acestor transformări se poate face prin creșterea dimensionalității sistemului de coordonate cartezian, definindu-se un sistem de coordonate cu 4 dimensiuni, numit sistem de coordonate omogene.

### 3.1.2 SISTEME DE COORDONATE OMOGENE

Se observă că transformările de scalare și de rotație se reprezintă prin înmulțiri de matrice, iar translația se reprezintă prin însumare de matrice. Reprezentarea unitară și combinarea transformărilor geometrice spațiale se poate face într-un sistem de coordonate cu patru dimensiuni, numit sistem de coordonate omogene. Un punct  $P(x, y, z)$  se reprezintă în sistemul de coordonate omogene ca  $P(X, Y, Z, w)$ , unde  $X = x \cdot w$ ,  $Y = y \cdot w$ ,  $Z = z \cdot w$ , pentru orice factor de scară  $w \neq 0$ .

În general, un punct  $P(x, y, z)$  în sistemul de coordonate cartezian se transformă în sistemul de coordonate omogene prin alegerea  $w = 1$ , deci are coordonatele omogene  $P(x, y, z, 1)$ . Pentru transformarea inversă, se calculează coordonatele carteziene ale unui punct  $P(X, Y, Z, w)$ , reprezentat în sistemul de coordonate omogene printr-o împărțire cu factorul de scară  $w$  astfel:

$$\begin{cases} x = X / w \\ y = Y / w \\ z = Z / w \end{cases} \quad (3.1)$$



În coordonate omogene, două puncte  $P_1(X_1, Y_1, Z_1, w_1)$  și  $P_2(X_2, Y_2, Z_2, w_2)$  sunt egale dacă :

$$X_1 / w_1 = X_2 / w_2, \quad Y_1 / w_1 = Y_2 / w_2 \text{ și } Z_1 / w_1 = Z_2 / w_2$$

În sistemele de coordonate omogene, toate matricile de transformări sunt de dimensiune  $4 \times 4$  și toate operațiile de transformări geometrice se pot exprima prin produse de matrice.

**Translația** în coordonate omogene se definește printr-o matrice:

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

Transformarea de translație a unui punct  $P(X, Y, Z, w)$  reprezentat în coordonate omogene, în punctul  $P'(X', Y', Z', w')$  se exprimă ca un produs de matrice:

$$P' = TP$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{ rezultă: } \begin{cases} X' = X + wt_x \\ Y' = Y + wt_y \\ Z' = Z + wt_z \\ w' = w \end{cases} \quad (3.3)$$

Teoretic, pentru a se obține coordonatele cartezine ale punctului transformat  $P'$ , trebuie să se execute operația de împărțire cu factorul de scară  $w'$  pentru fiecare coordonată:

$$\begin{cases} x' = X' / w' \\ y' = Y' / w' \\ z' = Z' / w' \end{cases}$$

Dar, dacă se alege  $w = 1$ , rezultă  $w' = 1$  și împărțirea nu mai este necesară. În general, transformările geometrice primitive conservă valoarea factorului de scară și, dacă se alege  $w = 1$ , împărțirea cu  $w'$  (numită împărțirea omogenă) nu este necesară.

**Scalarea față de origine** se reprezintă în sistemul de coordonate omogene prin matricea:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$



Transformarea de scalare a unui punct reprezentat în coordonate omogene  $P(X, Y, Z, w)$ , în punctul  $P'(X', Y', Z', w')$  este dată de relațiile:

$$P' = SP$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{rezultă: } \begin{cases} X' = s_x X \\ Y' = s_y Y \\ Z' = s_z Z \\ w' = w \end{cases} \quad (3.5)$$

Dacă factorii de scalare sunt egali ( $s_x = s_y = s_z$ ), scalarea se numește uniformă, și păstrează forma obiectului. Dacă factorii de scalare diferă, obiectul este deformat, iar scalarea se numește neuniformă.

**Transformările de rotație** în raport cu axele sistemului de referință exprimate în coordonate omogene sunt următoarele:

Rotația în raport cu axa  $x$  (tangaj) cu un unghi  $\theta$ :

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

În această transformare, axa  $x$  rămâne nemodificată, iar toate celelalte puncte din spațiu se transformă prin înmulțire cu matricea  $R_x(\theta)$ :

$$P' = R_x P$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{rezultă } \begin{cases} X' = X \\ Y' = Y \cos \theta - Z \sin \theta \\ Z' = Y \sin \theta + Z \cos \theta \\ w' = w \end{cases} \quad (3.7)$$

Rotația în raport cu axa  $y$  (girație) cu un unghi  $\chi$ :

$$R_y(\chi) = \begin{bmatrix} \cos \chi & 0 & \sin \chi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \chi & 0 & \cos \chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.8)$$

În această transformare, axa  $y$  rămâne nemodificată, iar toate celelalte puncte din spațiu se transformă prin înmulțire cu matricea  $R_y(\chi)$ :

$$P' = R_y P$$



$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos \chi & 0 & \sin \chi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \chi & 0 & \cos \chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{rezultă: } \begin{cases} X' = X \cos \chi + Z \sin \chi \\ Y' = Y \\ Z' = -X \sin \chi + Z \cos \chi \\ w' = w \end{cases} \quad (3.9)$$

Rotația în raport cu axa z (ruliu), cu un unghi  $\rho$  :

$$\mathbf{R}_z(\rho) = \begin{bmatrix} \cos \rho & -\sin \rho & 0 & 0 \\ \sin \rho & \cos \rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

În această transformare, axa z rămâne nemodificată, iar toate celelalte puncte din spațiu se transformă prin înmulțire cu matricea  $\mathbf{R}_z(\rho)$ :

$$\mathbf{P}' = \mathbf{R}_z \mathbf{P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos \rho & -\sin \rho & 0 & 0 \\ \sin \rho & \cos \rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}; \text{rezultă: } \begin{cases} X' = X \cos \rho - Y \sin \rho \\ Y' = X \sin \rho + Y \cos \rho \\ Z' = Z \\ w' = w \end{cases} \quad (3.11)$$

Toate matricele de transformare de rotație sunt matrice ortogonale și ortonormate. Transformările geometrice elementare sunt transformări liniare, prin care liniile drepte și suprafețele plane sunt transformate în linii drepte, respectiv suprafețe plane. Din această cauză, pentru transformarea unui obiect tridimensional este suficient să se transforme toate vârfurile acestuia și să se păstreze relațiile topologice între vârfurile transformate, aceleași cu cele între vârfurile inițiale.

Transformările mai complexe ale obiectelor în spațiu se pot defini prin compunerea mai multor transformări primitive.

### 3.1.3 COMPUNEREA TRANSFORMĂRILOR GEOMETRICE

Compunerea mai multor transformări elementare pentru obținerea unei transformări complexe se obține prin executarea succesivă a produsului fiecărei matrice de transformare cu matricea de reprezentare a punctului inițial sau rezultat dintr-o transformare precedentă. De exemplu, în fig. 3.3 este prezentată o transformare compusă a unui obiect.

Obiectul inițial este un cub cu latura de două unități, amplasat cu centrul său în centrul sistemului de referință și laturile orientate în direcțiile axelor de coordonate. După o scalare cu factorii de scară 2, 2, 2, o rotație cu un unghi de 30 grade în raport cu axa z și o translație cu un vector de translație cu componente 8,0,0, se obține un nou obiect cub, definit în același sistem de referință, dar cu alte dimensiuni și localizare. Reprezentarea din fig. 3.3 conține și o transformare de



proiecție perspectivă, pentru percepția adâncimii (a distanței față de observator a obiectelor), care va fi explicată ulterior. Una din fețele cubului (cea mai apropiată de observator) este desenată ca suprafață de culoare gri; celelalte fețe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare numită "cadru de sârmă" – *wireframe*).

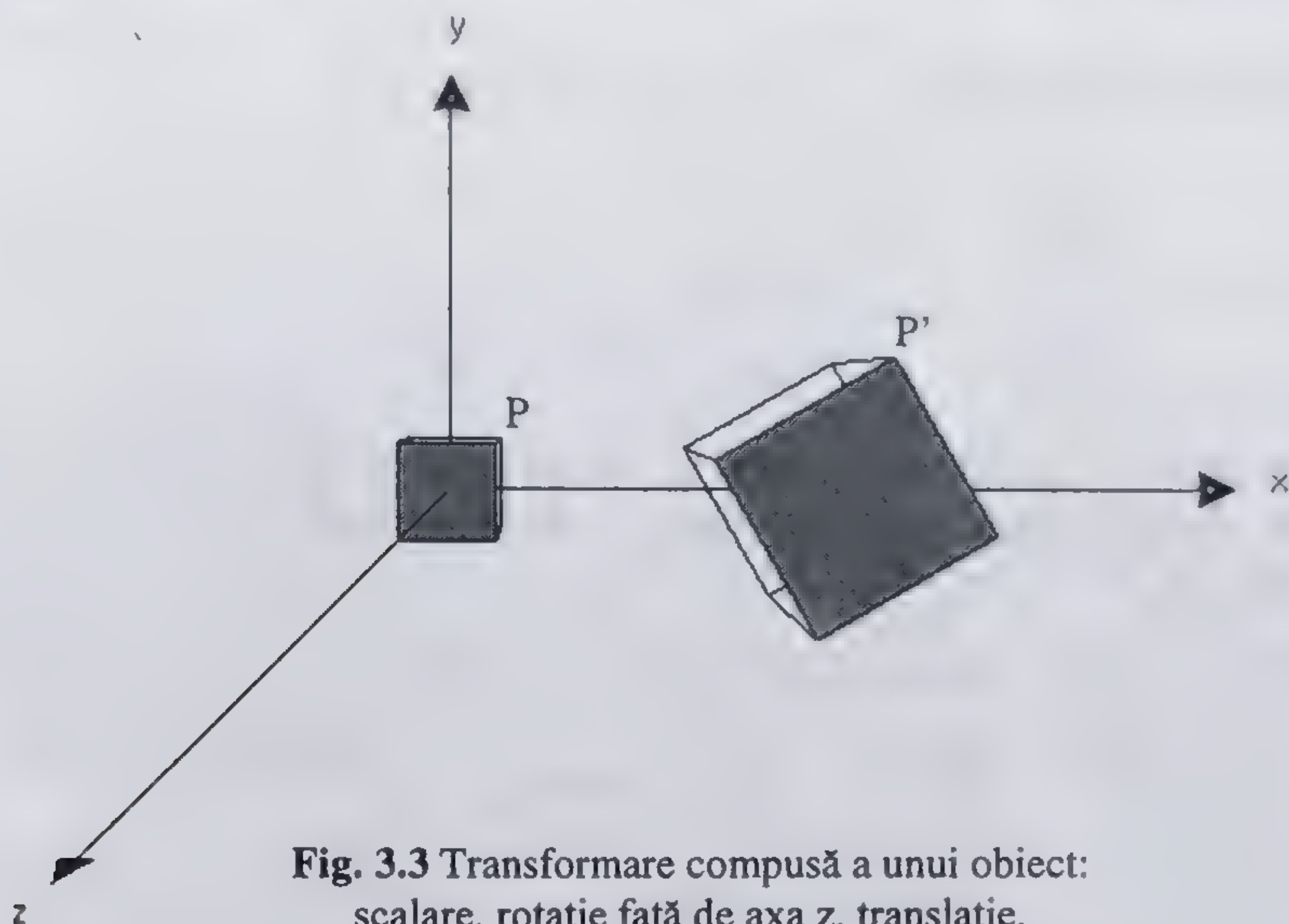


Fig. 3.3 Transformare compusă a unui obiect: scalare, rotație față de axa z, translație.

Transformarea efectuată asupra cubului din figura de mai sus se obține prin aplicarea succesivă a trei transformări geometrice elementare (scalare, rotație și translație) asupra fiecărui punct (vârf) al cubului. Pentru un vârf al cubului, reprezentat prin matricea coloană  $P$ , succesiunea de transformări este:

Scalarea:  $P_1 = S P$

Rotația în raport cu axa z:  $P_2 = R_z P_1 = R_z(S P)$

Translația:  $P' = T P_2 = T(R_z(S P)) = (T R_z S) P$

Nu este necesar să fie executate pe rând operațiile de înmulțire cu matricele de transformare  $S$ ,  $R_z$ , și  $T$ , ci se poate calcula o matrice compusă a transformării  $M$ , care se aplică apoi fiecărui punct  $P$  al obiectului:

$$P' = T R_z S P = M P, \text{ unde } M = T R_z S$$

O transformare compusă se poate deci defini printr-o matrice de transformare  $M$  care este un produs (compunere) de matrice de transformări geometrice elementare. În exemplul dat, se calculează matricea  $M$  astfel:

$$M = T R_z S = \begin{bmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0,866 & -0,5 & 0 & 0 \\ 0,5 & 0,866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$M = \begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1.732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unul din vârfurile cubului, vârful  $P(1,1,1,1)$  se transformă în  $P'(8.732, 3.732, 1, 1)$ :

$$P' = MP = \begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1.732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 8.732 \\ 3.732 \\ 2 \\ 1 \end{bmatrix}$$

Ordinea de compunere a matricelor de transformare este definitorie pentru rezultatul transformării, dat fiind că produsul matricelor nu este comutativ.

Se poate verifica pe exemplul dat, prin inversarea ordinii transformării de rotație cu translația (fig. 3.4).

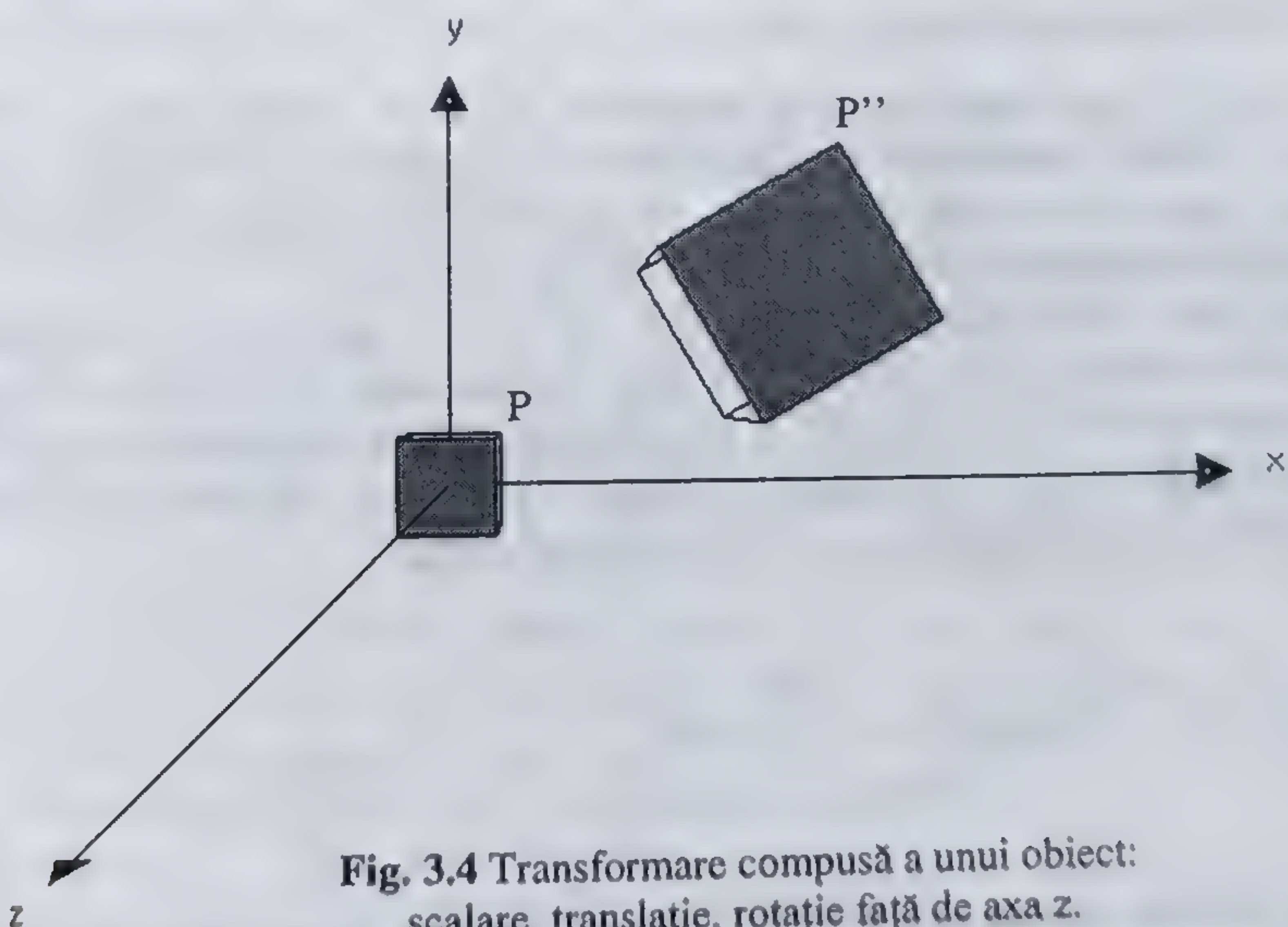


Fig. 3.4 Transformare compusă a unui obiect: scalare, translație, rotație față de axa z.

Se obține o matrice de transformare  $M''$  diferită de  $M$  și punctul transformat corespunzător  $P''$ , diferit de  $P'$ :

$$M'' = R_z TS = \begin{bmatrix} 1.732 & -1 & 0 & 6.928 \\ 1 & 1.732 & 0 & 4.330 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; P'' = M''P = \begin{bmatrix} 7.560 \\ 7.06 \\ 2 \\ 1 \end{bmatrix}$$



Convenția de reprezentare a punctelor în spațiu prin matrice coloană impune ordinea de înmulțire numită postmultiplicare (sau multiplicare la dreapta) a matricelor de transformare: se înmulțește matricea de transformare curentă cu matricea transformării următoare. Convenția de terminologie (postmultiplicare sau multiplicare la dreapta) semnifică faptul că o nouă transformare se concatenează ca factor dreapta al produsului de matrice. Este important de reținut faptul că ordinea de aplicare a transformărilor este de la dreapta la stânga din succesiunea de matrice ale unei compuneri. Mai precis, dacă un punct se transformă prin aplicarea succesivă a transformărilor definite prin matricele  $M_1, M_2, \dots, M_n$ , matricea compusă de transformare este:

$$M = M_n \cdots M_2 M_1 \quad (3.12)$$

Se poate verifica că, dacă se adoptă convenția de reprezentare a unui punct în spațiul tridimensional printr-o matrice linie, atunci transformarea unui punct se obține prin înmulțirea vectorului de poziție al punctului cu matricea de transformare (premultiplicare sau multiplicare la stânga). În această situație, ordinea în care se aplică matricele componente ale unei transformări compuse este de la stânga la dreapta:

$$P' = PS, \text{ unde } S = S_1 S_2 \cdots S_n \quad (3.13)$$

Punctul  $P$  este transformat în ordinea  $S_1, S_2, \dots, S_n$ . Acesta este un mod mai natural de a urmări secvențele de transformări geometrice și de aceea, în unele lucrări este adoptată această convenție. Nu există alte diferențe între cele două convenții (nici de simplificare a calculelor, nici de eficiență).

Se poate demonstra cu ușurință că orice matrice de transformare elementară în convenția de reprezentare prin matrice linie a unui punct în spațiul tridimensional este transpusa matricei corespunzătoare de transformare definite în convenția de reprezentare prin matrice coloană a punctului. De exemplu, translația unui punct  $P(x, y, z)$  cu valorile  $t_x, t_y, t_z$  se exprimă astfel:

$$[X' \ Y' \ Z' \ w'] = [X \ Y \ Z \ w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

În această lucrare se folosește convenția de reprezentare prin matrice coloană a punctelor din spațiu, dat fiind că aceasta este convenția din biblioteca grafică OpenGL, care este folosită pentru exemplificarea operațiilor grafice prezentate.

Un alt exemplu de transformare compusă este transformarea de rotație completă specificată prin trei rotații față de axele sistemului de coordonate. Cea mai obișnuită convenție pentru ordinea de specificare a rotațiilor este: ruliu cu unghiul  $\rho$  (după axa  $z$ ), tangaj cu unghiul  $\theta$  (după axa  $x$ ) și giratie cu unghiul  $\chi$  (față de axa  $y$ ). În această situație matricea de rotație totală  $R$  are expresia:



$$\mathbf{R} = \mathbf{R}_Y(\chi)\mathbf{R}_X(\theta)\mathbf{R}_Z(\rho) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

Matricea  $\mathbf{R}$  rezultată prin compunerea (înmulțirea) mai multor matrice de rotație este de asemenea ortogonală și ortonormată.

O transformare complexă a unui obiect prin combinarea mai multor transformări elementare (scalări, rotații, translații) se poate exprima printr-o matrice de transformare  $\mathbf{M}$  care are forma generală:

$$\mathbf{M} = \begin{bmatrix} r'_{11} & r'_{12} & r'_{13} & t_x \\ r'_{21} & r'_{22} & r'_{23} & t_y \\ r'_{31} & r'_{32} & r'_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.15)$$

Submatricea stânga-sus  $\mathbf{R}'$  de dimensiune  $3 \times 3$  exprimă transformarea de rotație și scalare totală, iar submatricea coloană dreapta  $\mathbf{T}$  exprimă transformarea de translație totală. Matricea  $\mathbf{R}'$  este o matrice ortogonală, dar poate să nu fie și ortonormată.

Două proprietăți importante ale transformărilor geometrice compuse trebuie să fie remarcate și reținute.

Prima proprietate este că o transformare geometrică compusă reprezentată printr-o matrice de forma (3.15) conservă valoarea coordonatei  $w$  a unui punct (acest lucru este evident). Această proprietate este folosită în implementarea operațiilor grafice, prin amânarea împărțirii omogene (cu  $w$ ) după ce s-au selectat numai obiectele vizibile (după operația de decupare).

A doua proprietate este proprietatea de liniaritate a transformărilor geometrice compuse, prin care liniile drepte și suprafețele plane sunt transformate în linii drepte, respectiv suprafețe plane. De aceea, la fel ca și în cazul transformărilor geometrice elementare, pentru transformarea unui obiect tridimensional este suficient să se transforme toate vârfurile acestuia și să se păstreze relațiile topologice între vârfurile transformate.

### 3.1.4 TRANSFORMĂRI INVERSE

Fiind dată o transformare a unui punct  $P$  într-un punct  $P'$  definită printr-o matrice de transformare  $\mathbf{M}$ , transformarea inversă, de la punctul  $P'$  la punctul  $P$  se obține prin înmulțirea cu matricea inversă,  $\mathbf{M}^{-1}$ :

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}, \quad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.16)$$



unde  $I$  este matricea identitate.

Dat fiind că, în general, matricea de transformare  $M$  se obține printr-un produs de matrice de transformări elementare, matricea inversă  $M^{-1}$  se calculează prin produsul în ordine inversă a inverselor matricelor elementare componente:

$$\begin{aligned} M &= M_n \cdots M_2 M_1 \\ M^{-1} &= M_1^{-1} M_2^{-1} \cdots M_n^{-1} \\ MM^{-1} &= M_n \cdots M_2 M_1 M_1^{-1} M_2^{-1} \cdots M_n^{-1} = I \end{aligned} \quad (3.17)$$

Relația (3.17) se demonstrează imediat, prin gruparea factorilor începând cu  $M_1 M_1^{-1} = I$ .

Toate matricele de transformări elementare sunt matrice inversabile și au următoarele expresii:

$$[T(t_x, t_y, t_z)]^{-1} = T(-t_x, -t_y, -t_z) = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.18)$$

$$[S(s_x, s_y, s_z)]^{-1} = S(1/s_x, 1/s_y, 1/s_z) = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.19)$$

$$[R_X(\theta)]^{-1} = R_X(-\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.20)$$

$$[R_Y(\chi)]^{-1} = R_Y(-\chi) = \begin{bmatrix} \cos \chi & 0 & -\sin \chi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \chi & 0 & \cos \chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.21)$$

$$[R_Z(\rho)]^{-1} = R_Z(-\rho) = \begin{bmatrix} \cos \rho & \sin \rho & 0 & 0 \\ -\sin \rho & \cos \rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.22)$$

$$R^{-1} = [R_Y(\chi) R_X(\theta) R_Z(\rho)]^{-1} = R_Z(-\rho) R_X(-\theta) R_Y(-\chi) \quad (3.23)$$

Datorită faptului că matricea de rotație este ortogonală și ortonormată, inversa unei matrice de rotație este egală cu transpusa acesteia.



### 3.1.5 TRANSFORMAREA SISTEMELOR DE REFERINȚĂ

Interpretarea relațiilor de transformări geometrice prezentate până acum a fost aceea de manevrare și modificare a obiectelor într-un sistem de referință dat: obiectele sunt reprezentate într-un anumit sistem de referință prin coordonatele unei mulțimi de puncte ale acestora (vârfuri) și ele pot fi deplasate, reorientate sau redimensionate prin aplicarea transformărilor geometrice corespunzătoare.

O altă interpretare care se poate da operațiilor de transformări geometrice este aceea de schimbare a sistemului de referință.

Se consideră sistemul de referință  $Oxyz$  și un nou sistem de referință  $O'x'y'z'$ , a cărui origine  $O'$  este determinată în sistemul  $Oxyz$  prin coordonatele  $x_0, y_0, z_0$  ale centrului  $O'$ . Sistemul de referință  $Oxyz$  este definit de versorii (vectori unitate)  $\mathbf{i}, \mathbf{j}, \mathbf{k}$ , iar sistemul de referință  $O'x'y'z'$  de versorii  $\mathbf{i}', \mathbf{j}', \mathbf{k}'$ .

Axa  $O'x'$  are cosinușii directori  $c_{11}, c_{12}, c_{13}$  față de sistemul de referință  $Oxyz$ ; axa  $O'y'$  are cosinușii directori  $c_{21}, c_{22}, c_{23}$  față de sistemul de referință  $Oxyz$ ; axa  $O'z'$  are cosinușii directori  $c_{31}, c_{32}, c_{33}$  față de sistemul de referință  $Oxyz$ .

Matricea de transformare care descrie poziționarea sistemului  $O'x'y'z'$  relativ la sistemul de referință  $Oxyz$  este:

$$\mathbf{M} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & x_0 \\ c_{21} & c_{22} & c_{23} & y_0 \\ c_{31} & c_{32} & c_{33} & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.24)$$

Fie un punct  $P$  definit prin coordonatele sale  $x, y, z$  în sistemul de referință  $Oxyz$ . Se demonstrează [Drag57], că în sistemul de referință  $O'x'y'z'$  acest punct (notat  $P'$ ) are coordonatele  $x', y', z'$ , care se obțin prin înmulțirea matricei  $\mathbf{M}^{-1}$  (care este inversa matricei  $\mathbf{M}$  care definește poziționarea sistemului  $O'x'y'z'$  în sistemul  $Oxyz$ ) cu matricea  $\mathbf{P}$  de reprezentare a punctului  $P$  în sistemul de coordonate inițial:

$$\mathbf{P}' = \mathbf{M}^{-1} \mathbf{P} \quad (3.25)$$

Transformarea inversă, a unui punct  $P'(x', y', z')$  din sistemul de referință  $O'x'y'z'$  în punctul  $P(x, y, z)$  din sistemul de referință  $Oxyz$  se obține prin înmulțire cu matricea de transformare  $\mathbf{M}$ :

$$\mathbf{P} = \mathbf{M} \mathbf{P}' \quad (3.26)$$

Se poate urmări cu mai multă ușurință această modalitate de transformare într-un caz simplu. Se consideră un sistem de referință  $Oxyz$  și un alt sistem de referință  $O'x'y'z'$  care are originea  $O'(x_0, y_0, z_0)$  și aceeași orientare a axelor de coordonate ca și sistemul  $Oxyz$ .

Poziționarea sistemului de referință  $O'x'y'z'$  relativ la sistemul  $Oxyz$  este definită de matricea de transformare  $\mathbf{M}$ ; poziționarea sistemului  $Oxyz$  relativ la sistemul  $O'x'y'z'$  este definită de matricea de transformare inversă  $\mathbf{M}^{-1}$ .



$$M = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; M^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Submatricea de rotație componentă a matricei  $M$  este matricea unitate, dat fiind că sistemul de referință  $O'x'y'z'$  are axele cu aceeași orientare ca și sistemul de referință. Într-adevăr, cosinușii directori ai axelor  $O'x', O'y', O'z'$  față de axele sistemului de referință  $Oxyz$  sunt  $(1,0,0)$ ,  $(0,1,0)$ ,  $(0,0,1)$  și matricea de rotație este matricea unitate.

Un punct oarecare  $P(x,y,z)$  în sistemul de referință  $Oxyz$ , se transformă în punctul  $P'(x',y',z')$  în sistemul de referință  $O'x'y'z'$  printr-o translație cu  $(-x_0, -y_0, -z_0)$  deci :

$$P' = M^{-1}P; \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Particularizarea pentru  $O'(4,2,0)$  și  $P(6,3,0)$  este reprezentată în fig. 3.5. Punctul  $P'(2,1,0)$  este transformatul punctului  $P$  în sistemul de referință  $O'x'y'z'$ .

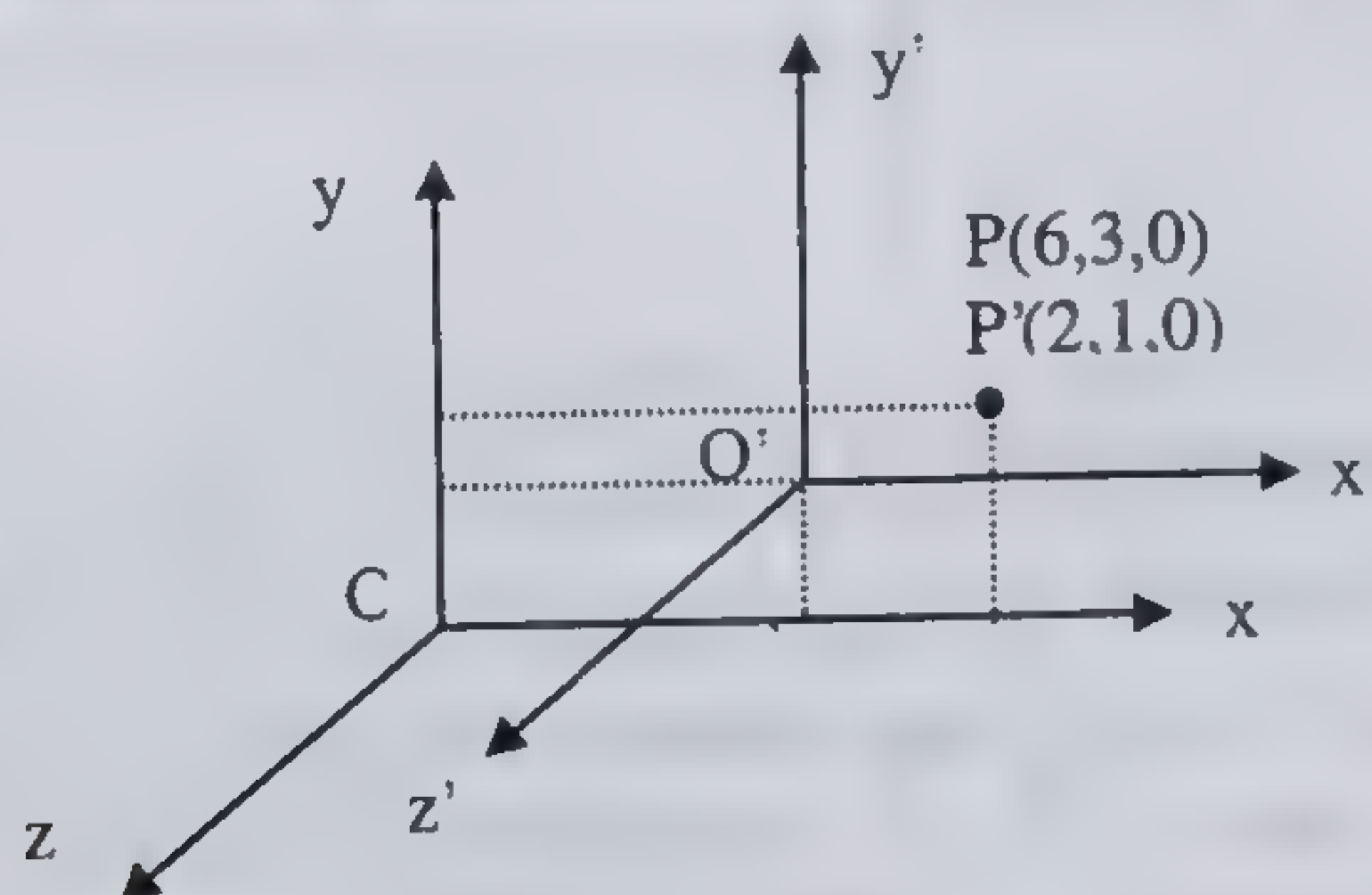


Fig.3.5 Transformarea sistemelor de referință.

În concluzie, aplicarea unei transformări definite printr-o matrice  $M$ , asupra unei mulțimi de puncte definite într-un sistem de referință  $Oxyz$  poate fi interpretată în mai multe moduri:

- Fiecare punct este modificat și capătă o nouă poziționare în sistemul de referință  $Oxyz$ , conform cu matricea de transformare  $M$ .
- Fiecare punct este transformat din sistemul de referință inițial  $Oxyz$  într-un nou sistem de referință,  $O'x'y'z'$ , a cărui poziție și orientare relativ la sistemul de referință  $Oxyz$  este descrisă de matricea  $M^{-1}$ .
- Fiecare punct este transformat din sistemul de referință inițial  $Oxyz$  într-un nou sistem de referință,  $O'x'y'z'$ . Poziția și orientarea sistemului  $Oxyz$  relativ la sistemul de referință  $O'x'y'z'$  este descrisă de matricea  $M$ .



Și încă o precizare: un sistem de referință nu este “materializat” în nici un fel într-un sistem grafic (în software sau hardware), ci este o convenție cunoscută de programator sau utilizator. Trecerea de la un sistem de referință la altul se efectuează prin transformări aplicate punctelor (obiectelor): modificarea coordonatelor acestora le transferă dintr-un sistem de referință în altul. În acest sens, orice transformare geometrică poate fi considerată ca o schimbare a sistemului de referință. Dar, după cum se va observa în continuare, transformările geometrice sunt uneori considerate ca modificări ale obiectelor, iar în alte situații, ca schimbare a sistemului de referință. Este normal ca un începător în practica programării sistemelor grafice să pună o justificată întrebare: cum se poate ști care este interpretarea corectă a unei transformări? Răspunsul este că ambele interpretări sunt corecte, dar se alege aceea care permite urmărirea cea mai directă a unui raționament pentru calculul unor transformări complexe. Nu există “rețete” unice și sigure care să poată fi aplicate fără greșală în orice situație, deci experiența de proiectare își spune întotdeauna cuvântul.

În dezvoltarea sistemelor de vizualizare se folosesc, totuși, câteva sisteme de referință bine definite, care permit specificarea cea mai simplă și eficientă a operațiilor de redare a obiectelor (scenelor). Unul dintre acestea, sistemul de referință universal, a fost deja introdus. Alte sisteme de referință folosite în grafica tridimensională vor fi definite pe parcursul lucrării. Sistemele de referință intermediare, care apar în descrierea unor transformări complexe, sunt interpretări ale operațiilor de transformări geometrice care permit urmărirea unui raționament de calcul. Exemplul următor evidențiază acest aspect.

### ■ Exemplul 3.1

Se consideră sistemul de coordonate Oxyz și o matrice de transformare  $R_x(\pi/2)$  care realizează o rotație cu unghiul  $\pi/2$  în raport cu axa x:

$$R_x(\pi/2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Această transformare poate fi interpretată ca o transformare prin rotație cu  $90^\circ$  în raport cu axa x a fiecărui punct  $P(x,y,z)$ , în punctul transformat  $P'(x',y',z')$ . În fig.3.6 (a) este considerat punctul  $P(0,1,1)$ , care se transformă în punctul  $P'(0,-1,1)$ . În fig. 3.6(b) este reprezentată transformarea echivalentă a sistemului de coordonate Oxyz în sistemul de coordonate  $O'x'y'z'$  folosind matricea de transformare inversă corespunzătoare,  $R_x(-\pi/2)$ : axa y se transformă în axa  $z'$ , iar axa z se transformă în axa  $-y'$ . Punctul P, de coordonate  $(0,1,1)$  în sistemul Oxyz, se transformă în punctul  $P'$  de coordonate  $(0,-1,1)$  în sistemul  $O'x'y'z'$ .

Se pot verifica ușor rotațiile cu un unghi  $\pi/2$  în raport cu celelalte axe de coordonate. Rotația cu  $\pi/2$  față de axa y transformă sistemul de coordonate într-un sistem în care axa z se schimbă în axa x, iar axa x se schimbă în axa  $-z$ . Rotația cu



$\pi/2$  față de axa  $z$  transformă sistemul de coordonate într-un sistem în care axa  $x$  se schimbă în axa  $y$ , iar axa  $y$  se transformă în axa  $-x$ .

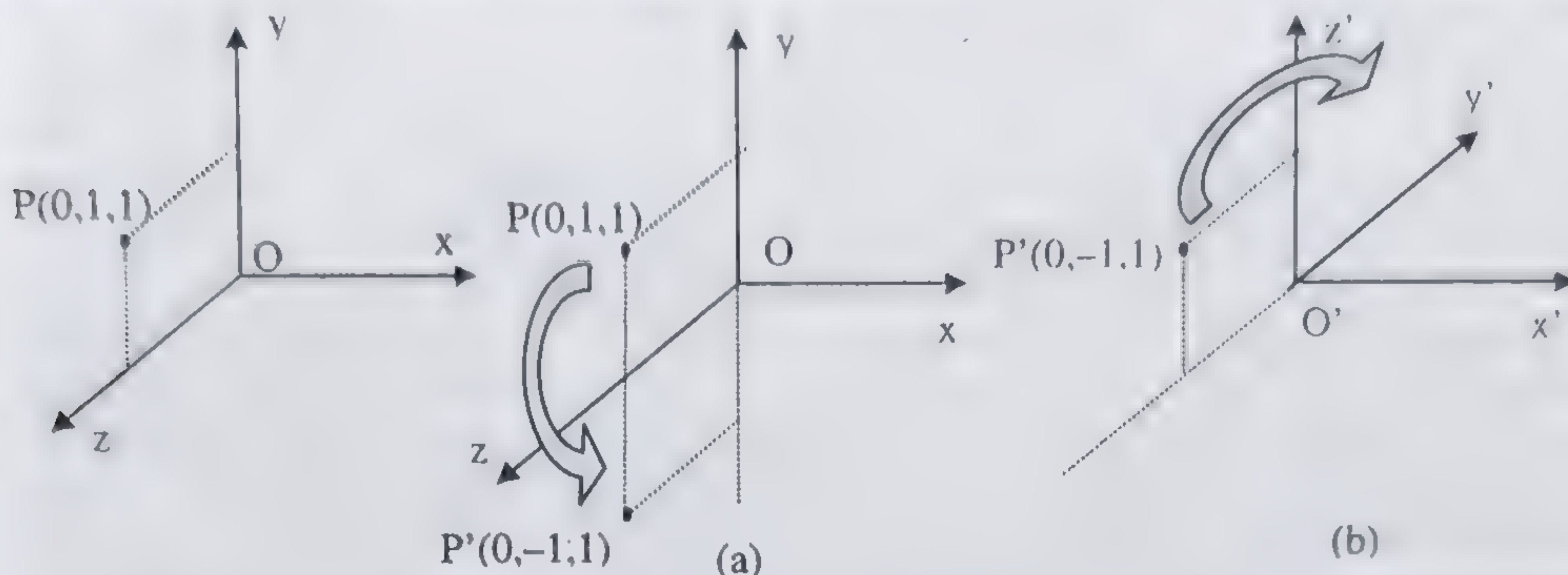


Fig. 3.6 Rotație cu  $\pi/2$  în raport cu axa  $x$ :

(a) rotația punctelor cu  $\pi/2$ ; (b) rotația sistemului de coordonate cu  $-\pi/2$ .

Pentru deducerea matricelor unor transformări complexe se procedează prin reducerea acestora la cazuri mai simple, pentru care se cunosc expresiile de calcul, folosind transformări primitive succesive. Inițial se aplică o transformare ajutătoare pentru aducerea obiectelor într-o poziție sau sistem de referință adecvat, se execută transformarea dorită, după care se revine la poziția sau sistemul de referință inițial prin transformarea inversă celei aplicate inițial. Câteva exemple prezentate în continuare vor preciza acest mod de realizare a transformărilor geometrice compuse.

### 3.1.5.1 Rotația față de o axă paralelă cu una din axele sistemului de referință

Un exemplu ilustrativ de compunere a transformărilor geometrice este calculul rotației în raport cu o axă paralelă cu una din axele sistemului de referință.

Se consideră o dreaptă  $D$  paralelă cu axa  $z$  a sistemului, care intersectează planul  $Oxy$  în punctul  $I(t_x, t_y, 0)$ . Transformarea de rotație a obiectelor față de această axă nu poate fi realizată folosind matricea dedusă în paragraful precedent (relația 3.10), care presupune rotația față de o axă a sistemului de coordonate.

De aceea, se aplică mai întâi tuturor punctelor o transformare ajutătoare, translația cu  $T(-t_x, -t_y, 0)$ . Această transformare poate fi interpretată în două moduri: ca o schimbare a sistemului de referință  $Oxyz$  în sistemul  $O'x'y'z'$ , în care punctul  $I$  este transformat în punctul  $I'(0,0,0)$ , sau ca o modificare a poziției tuturor punctelor din spațiu prin care punctul  $I$  este adus în originea sistemului de referință  $Oxyz$ .

Dacă se consideră prima interpretare, operațiile se continuă în noul sistem de referință  $O'x'y'z'$ , în care dreapta  $d$  se suprapune peste axa  $z'$ , deci se poate obține rotația dorită cu un unghi  $\rho$  folosind matricea de rotație (3.10). După aceasta, se revine la sistemul de referință inițial  $Oxyz$ , printr-o transformare



inversă,  $T(t_x, t_y, 0)$ . Rezultă matricea compusă de rotație cu unghiul  $\rho$  față de o dreaptă paralelă cu axa  $z$  care intersectează planul  $Oxy$  în punctul  $I(t_x, t_y, 0)$ :

$$R_D = T(t_x, t_y, 0)R_Z(\rho)T(-t_x, -t_y, 0)$$

$$R_D = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \rho & -\sin \rho & 0 & 0 \\ \sin \rho & \cos \rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_D = \begin{bmatrix} \cos \rho & -\sin \rho & 0 & t_x(1 - \cos \rho) + t_y \sin \rho \\ \sin \rho & \cos \rho & 0 & t_y(1 - \cos \rho) - t_x \sin \rho \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.27)$$

Acest proces este descris în fig. 3.7 (a) pentru un unghi de 30 grade, cu reprezentarea unei proiecții în planul  $Oxy$ .

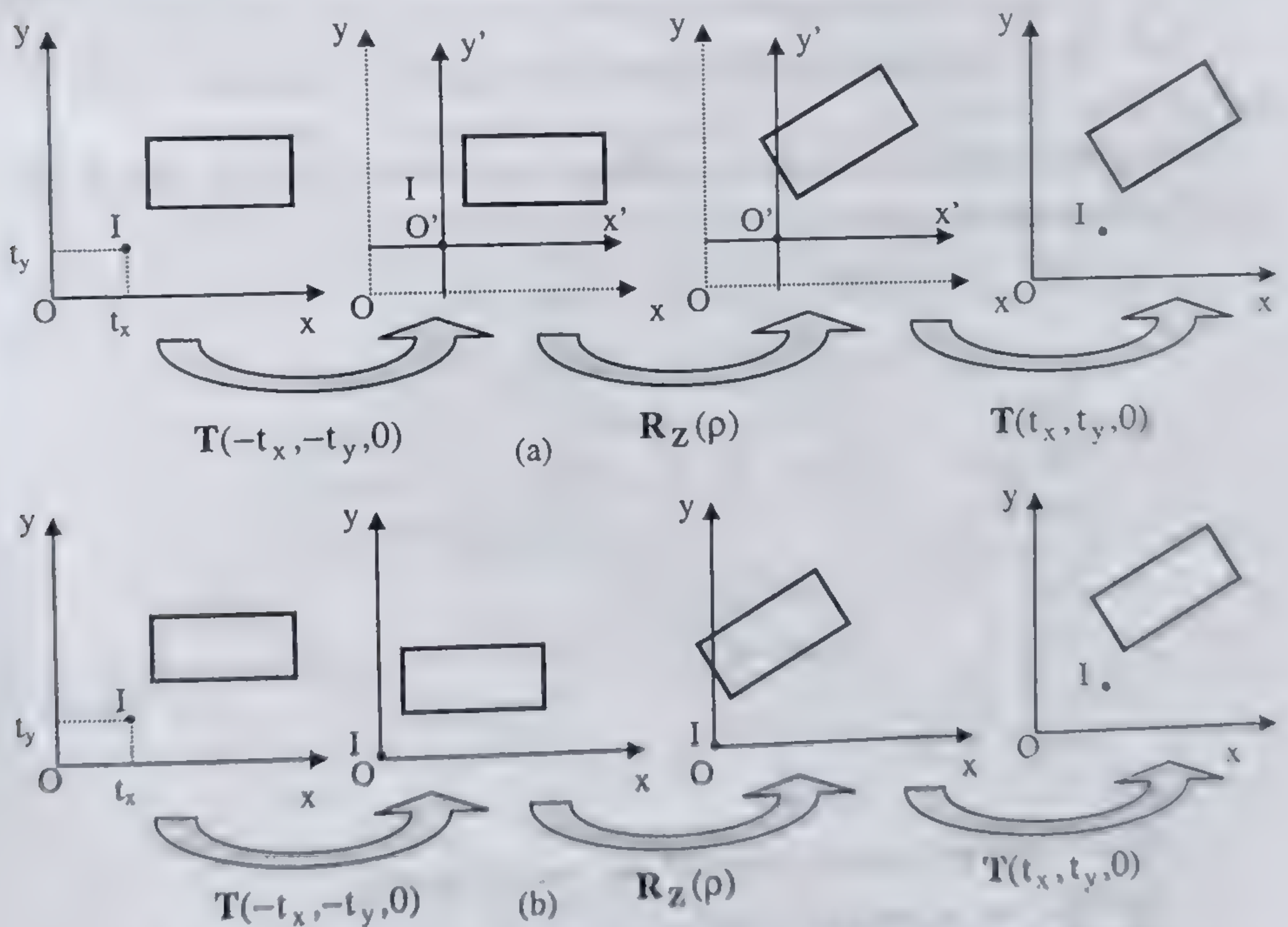


Fig. 3.7 Rotația față de o dreaptă paralelă cu axa  $z$  a sistemului de referință:  
(a) considerând transformarea sistemelor de referință;  
(b) considerând transformarea punctelor.

În fig. 3.7 (b) se prezintă aceleași transformări succesive necesare pentru realizarea rotației în raport cu o dreaptă paralelă cu axa  $z$  a sistemului, considerând



că transformarea inițială  $T(-t_x, -t_y, 0)$  transformă toate punctele din sistemul de referință  $Oxyz$  astfel încât punctul  $I$  se suprapune peste originea  $O$ . Se remarcă faptul că, în această situație, este doar o diferență de interpretare, operațiile și rezultatul acestora fiind identic.

### 3.1.5.2 Scalarea față de un punct oarecare în spațiu

Matricea de scalare din relația (3.4) efectuează scalarea față de originea sistemului de coordonate, prin care componentele  $x, y, z$  ale vectorului de poziție  $OP$  al punctului  $P$  sunt multiplicat fiecare cu factorul de scalare corespunzător. Se poate defini o scalare față de un punct fix  $F(x_f, y_f, z_f)$  din spațiu, prin care componentele vectorului  $FP$  sunt multiplicat cu factorii de scalare corespunzători.

Matricea de scalare față de un punct oarecare se poate deduce printr-o metodă asemănătoare celei prezentate anterior, prin compunerea a trei transformări elementare deja definite.

1. Se execută o translație cu  $T(-x_f, -y_f, -z_f)$ , prin care punctul fix  $F$  se suprapune peste originea sistemului de coordonate.
2. Se aplică o transformare de scalare cu matricea  $S(s_x, s_y, s_z)$ .
3. Se efectuează o translație inversă celei aplicate la punctul 1, deci cu matricea  $T(x_f, y_f, z_f)$ .

Rezultă matricea de scalare cu factorii de scară  $(s_x, s_y, s_z)$  față de un punct fix din spațiu  $F(x_f, y_f, z_f)$ :

$$S_F = T(x_f, y_f, z_f) S(s_x, s_y, s_z) T(-x_f, -y_f, -z_f)$$

$$S_F = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S_F = \begin{bmatrix} s_x & 0 & 0 & x_f(1-s_x) \\ 0 & s_y & 0 & y_f(1-s_y) \\ 0 & 0 & s_z & z_f(1-s_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.28)$$

### 3.1.5.3 Rotația față de o dreaptă oarecare în spațiu

Urmărind un raționament asemănător cu cele anterioare, se poate deduce matricea de rotație cu un unghi  $\delta$  față de o dreaptă oarecare  $D$  din spațiu, dată printr-un punct  $D_0(x_0, y_0, z_0)$  și vectorul unitate (versorul)  $\mathbf{d}$ , având cosinușii directori  $d_x, d_y, d_z$  ( $\mathbf{d} = d_x \mathbf{i} + d_y \mathbf{j} + d_z \mathbf{k}$ ). Pașii de transformare sunt următorii:

1. Translația cu matricea  $T(-x_0, -y_0, -z_0)$ , prin care punctul  $D_0$  ajunge în originea sistemului de referință.



2. Alinierea dreptei  $D$  cu una din axele sistemului de referință, de exemplu, axa  $z$ . Alinierea cu una din axele de coordonate ale sistemului se efectuează prin două rotații în raport cu celelalte două axe.

Mai întâi se efectuează o rotație în raport cu axa  $x$ , cu un unghi  $\alpha$ , astfel ca dreapta  $D$  să ajungă în planul  $Oxz$ . Pentru calculul unghiului  $\alpha$  se consideră notațiile din fig. 3.8 (b).  $OM$  este versorul dreptei  $D$  după translația efectuată în pasul precedent.  $M_x$ ,  $M_y$ ,  $M_z$  sunt intersecțiile planelor paralele cu planele sistemului de coordonate care trec prin punctul  $M$ , cu axele corespunzătoare ( $M_x$  este intersecția planului paralel cu planul  $Oyz$  care trece prin  $M$ , cu axa  $x$ , etc.).  $M_{xy}$ ,  $M_{yz}$ ,  $M_{zx}$  sunt proiecțiile punctului  $M$  pe planele  $O_{xy}$ ,  $O_{yz}$  și, respectiv,  $O_{zx}$ . Se pot scrie următoarele relații:

$$OM_x = d_x; OM_y = d_y; OM_z = d_z$$

$$MM_x = d_{yz} = \sqrt{d_y^2 + d_z^2}$$

$$\cos \alpha = d_z / d_{yz} = d_z / \sqrt{d_y^2 + d_z^2}; \sin \alpha = d_y / d_{yz} = d_y / \sqrt{d_y^2 + d_z^2} \quad (3.29)$$

Rotația în raport cu axa  $x$  se efectuează în planul  $MM_{xy}M_{zx}$ . Punctul  $M$ , rotit cu unghiul  $\alpha$  în planul  $MM_{xy}M_{zx}$ , se transformă în punctul  $M'$  aflat în planul  $O_{zx}$ , pe dreapta  $M_xM_{zx}$  paralelă cu axa  $Oz$ , astfel că  $MM_x = M'M_x = d_{yz}$ .

A doua etapă pentru alinierea versorului  $OM$  al dreptei  $D$  cu axa  $z$  este o rotație cu unghiul  $\beta$ , efectuată în planul  $Ozx$ , în raport cu axa  $y$ . Pentru calculul unghiului  $\beta$  se urmăresc notațiile din fig. 3.8 (c). În triunghiul dreptunghic  $OM'M_x$  (unghiul drept în  $M_x$ )  $OM_x = d_x$ ,  $OM' = 1$ ,  $M_xM' = d_{yz}$ . Rezultă unghiul  $OM'M_x$  (notat unghiul  $\beta$ ):

$$\cos \beta = d_x = \sqrt{d_y^2 + d_z^2}; \sin \beta = d_x \quad (3.30)$$

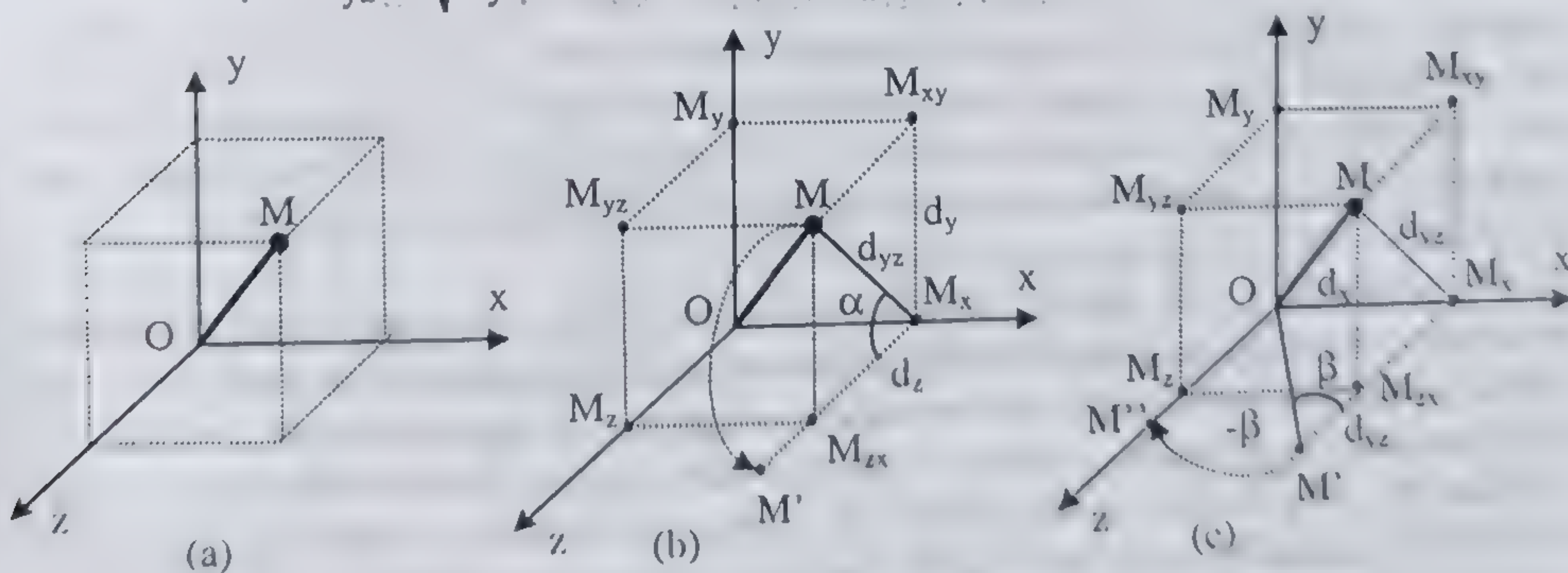


Fig. 3.8 Alinierea dreptei  $D$  cu axa  $z$ .

Rotația dreptei  $OM'$  în planul  $Ozx$  față de axa  $Oy$  transformă punctul  $M'$  în punctul  $M''$  pe axa  $Oz$  printr-o rotație cu unghiul de valoare  $\beta$  în sens invers sensului pozitiv de rotație după axa  $y$  (a se vedea fig. 3.3) Această transformare de aliniere a dreptei  $D$  cu axa  $z$  se exprimă matriceal astfel:

$$A_z = R_y(-\beta)R_x(\alpha)$$



3. În acest pas se execută rotația dorită, cu unghiul  $\delta$  în raport cu dreapta D, care este aliniată cu axa z a sistemului, deci se poate scrie:

$$\mathbf{R}_Z(\delta) = \begin{bmatrix} 1 & \cos \delta & -\sin \delta & 0 \\ 0 & \sin \delta & \cos \delta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. În pasul al patrulea se execută transformarea inversă celei de la pasul 2, deci cu o matrice  $\mathbf{A}_Z^{-1} = \mathbf{R}_X(-\alpha)\mathbf{R}_Y(\beta)$ .

5. În ultimul pas se execută transformarea de translație inversă față de cea executată în pasul 1, cu matricea  $\mathbf{T}(x_0, y_0, z_0)$ .

Rezultă matricea compusă de rotație față de dreapta D cu unghiul  $\delta$ :

$$\mathbf{R}_D = \mathbf{T}(x_0, y_0, z_0) \mathbf{R}_X(-\alpha) \mathbf{R}_Y(\beta) \mathbf{R}_Z(\delta) \mathbf{R}_Y(-\beta) \mathbf{R}_X(\alpha) \mathbf{T}(-x_0, -y_0, -z_0) \quad (3.31)$$

Valorile unghiurilor  $\alpha$  și  $\beta$  rezultă din cosinușii directori ai axei de rotație D și au valorile date de relațiile (3.29) și (3.30).

#### 3.1.5.4 Transformarea de modelare

O scenă virtuală este compusă dintr-un număr oarecare de obiecte tridimensionale amplasate în diferite poziții în scenă. Tehnica de modelare a scenelor de dimensiuni mari (ca număr de obiecte sau ca volum ocupat de scenă) se bazează pe amplasarea modelelor obiectelor tridimensionale în sistemul de referință universal al scenei virtuale.

Fiecare obiect este modelat într-un sistem de referință local (numit și sistem de referință de modelare), prin alegerea celei mai convenabile origini și orientări a axelor de coordonate ale acestuia. De exemplu, un cub se modelează cel mai convenabil într-un sistem de referință ales cu centrul în centrul cubului și cu axele orientate paralel cu laturile cubului. Dacă se presupune că latura cubului este egală cu 2 unități, atunci coordonatele vârfurilor cubului sunt  $(-1, -1, -1)$ ,  $(1, -1, -1)$ ,  $(1, 1, -1)$ ,  $(-1, 1, -1)$ ,  $(-1, -1, 1)$ ,  $(1, -1, 1)$ ,  $(1, 1, 1)$ ,  $(-1, 1, 1)$ .

Modelul unui obiect, definit în sistemul de referință local, poate fi instanțiat (introdus) în scena virtuală printr-o transformare numită transformare de modelare (sau transformare de instanțiere). Mai mult, reprezentarea ierarhică a scenelor virtuale prin grafuri aciclice direcționate permite instanțierea de un număr oarecare de ori a fiecărui model, folosind matrice de transformare corespunzătoare. Acesta este modul cel mai frecvent de modelare a scenelor virtuale, care are suport în formatele și limbajele de descriere a scenelor virtuale, cum este limbajul VRML.

În modelarea ierarhică a scenelor virtuale, operația de bază este transformarea de modelare prin care toate punctele unui obiect (model tridimensional) sunt transformate din sistemul de referință local într-un alt sistem de referință, care poate fi sistemul de referință universal sau un alt sistem intermediar. Pentru început se consideră cea mai simplă transformare de modelare, dintr-un sistem de referință local în sistemul de referință universal.



În mod obișnuit, prima transformare care se aplică modelului este transformarea de scalare cu o matrice de scalare  $S(s_x, s_y, s_z)$ , executată în sistemul de referință local (față de originea acestuia), prin care obiectul este adus la dimensiunile dorite ale instanței respective.

Poziția unei instanțe a obiectului în scenă se definește printr-o matrice de transformare conținând submatricea de rotație  $R$ , care definește orientarea sistemului de referință local  $O'x'y'z'$  față de sistemul de referință universal  $Oxyz$  (al scenei virtuale) și submatricea de translație  $T$ , care definește poziția originii  $O'(x_0, y_0, z_0)$  a sistemului de referință local în sistemul universal. Submatricea de rotație  $R$  poate fi specificată fie prin cosinuşii directori  $(c_{11}, c_{12}, c_{13})$ ,  $(c_{21}, c_{22}, c_{23})$ ,  $(c_{31}, c_{32}, c_{33})$  ai axelor sistemului local față de sistemul de referință universal, fie printr-o succesiune de rotații față de axele de coordonate ale sistemului de referință local:

$$R = \begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & c_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ sau } R = R_Y(\chi)R_X(\theta)R_Z(\rho)$$

Rezultă matricea de transformare de modelare (instanțiere):

$$M_I = T(x_0, y_0, z_0)RS(s_x, s_y, s_z) \quad (3.32)$$

Un punct  $P$  al obiectului în sistemul de referință model este transformat în punctul  $P_I$  în sistemul de referință universal prin instanțierea:

$$P_I = M_I P \quad (3.33)$$

Exemplul următor ilustrează crearea unei scene virtuale simple prin transformări de instanțiere.

### ■ Exemplul 3.2

Se consideră un cub ca latura de dimensiune 2, modelat într-un sistem de referință local ales cu centrul în centrul cubului și cu axele orientate paralel cu laturile cubului și se construiește o scenă virtuală (în sistemul de referință universal) în care se instanțiază patru obiecte pornind de la modelul cub.

Prima instanțiere este chiar cubul din sistemul de referință model, deci sistemul de referință model are același centru și aceeași orientare a axelor ca și sistemul de referință universal; deci  $C_1(0,0,0)$ .

A doua instanțiere este un cub cu latura de 4 unități, cu orientarea sistemului de referință local cu axele de coordonate paralele cu axele sistemului de referință al scenei, amplasat cu centrul în punctul  $C_2(0,-8,0)$ .

A treia instanțiere este un paralelipiped dreptunghic, cu laturile de dimensiuni 4,2,4, amplasat cu centrul în poziția  $C_3(8,0,0)$ . Sistemul de referință local are axa  $z'$  paralelă cu axa  $z$  a sistemului universal, iar axele  $x'$  și  $y'$  sunt rotite cu un unghi de  $\pi/4$  față de axa  $x$  și, respectiv,  $y$  a sistemului universal.



A patra instanțiere este un paralelipiped dreptunghic, cu laturile de dimensiuni 4,2,4, amplasat cu centrul în poziția  $C_4(-8,0,0)$ . Sistemul de referință local are axa  $z'$  paralelă cu axa  $z$  a sistemului universal, iar axele  $x'$  și  $y'$  sunt rotite cu un unghi de  $-\pi/4$  față de axa  $x$  și, respectiv,  $y$  a sistemului universal.

Pentru crearea scenei se calculează pe rând cele trei matrice de instanțiere.

Matricea de instanțiere  $M_1$  este chiar matricea identitate:  $M_1 = I$ .

Pentru calculul matricei de instanțiere  $M_2$ , se deduc în mod foarte simplu matricele  $S_2=S(2,2,2)$ ,  $R_2=I$ , și  $T_2=T(0,-8,0)$ , deci:  $M_2 = T(0,-8,0) S(2,2,2)$ .

Pentru calculul matricei de instanțiere  $M_3$ , se poate scrie  $S_3=S(2,1,2)$ ,  $R_3=R_Z(\pi/4)$ ,  $T_3=T(8,0,0)$ , deci:  $M_3 = T(8,0,0) R_Z(\pi/4) S(2,1,2)$ .

Pentru calculul matricei de instanțiere  $M_4$ , se poate scrie:  $S_4=S(2,1,2)$ ,  $R_4=R_Z(-\pi/4)$ ,  $T_4=T(-8,0,0)$ ,  $M_4 = T(-8,0,0) R_Z(-\pi/4) S(2,1,2)$ .

Scena virtuală rezultată este redată în fig. 3.9. Axele de coordonate reprezentate în imagine sunt axele sistemului de referință universal. În centrul sistemului de referință universal este reprezentat primul cub, cu aceleași dimensiuni ca cele ale modelului.

Cubul din partea de jos a imaginii se obține prin transformarea de instanțiere  $P_2 = M_2 P$ ; paralelipipedul din dreapta se obține prin transformarea de instanțiere  $P_3 = M_3 P$ ; paralelipipedul din stânga se obține prin transformarea de instanțiere  $P_4 = M_4 P$ .

Reprezentarea de mai jos conține și o transformare de proiecție perspectivă, pentru percepția adâncimii care va fi explicată ulterior. Una din fețele cubului (cea mai apropiată de observator) este desenată ca suprafață de o culoare gri; celelalte fețe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare "cadru de sârmă" – *wireframe*)

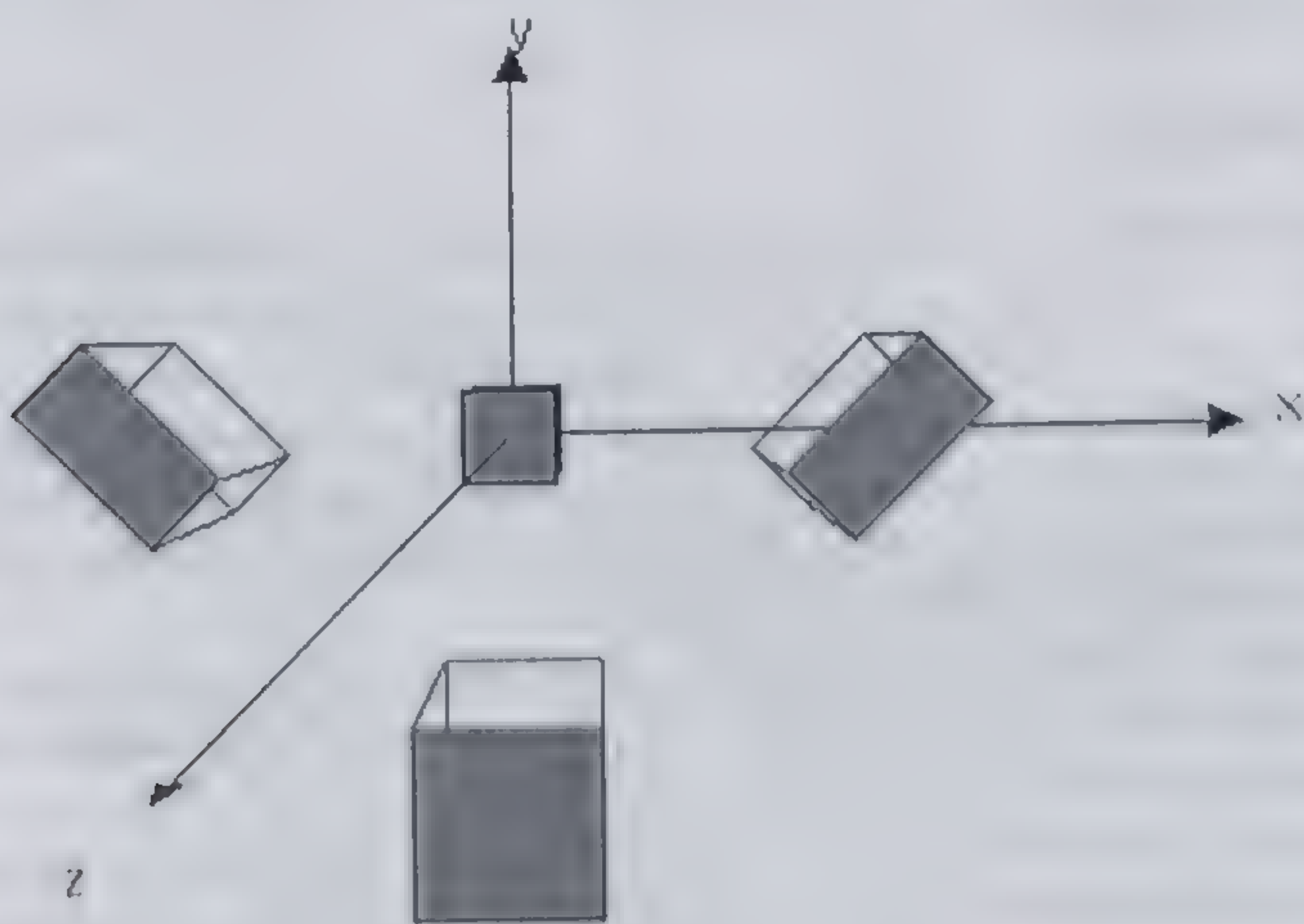


Fig. 3.9 Scenă virtuală rezultată prin instanțierea mai multor obiecte.



### 3.1.6 ALTE TRANSFORMĂRI GEOMETRICE ÎN SPAȚIU

Printre alte transformări în spațiul tridimensional care se pot aplica obiectelor, cele mai importante sunt transformarea de simetrie și transformarea de forfecare.

**Transformarea de simetrie.** O transformare de simetrie (oglindire) a unui punct  $P$  față de un plan  $\pi$  calculează un punct  $P'$  aflat pe o dreaptă  $\Delta$  perpendiculară pe planul  $\pi$ , la o distanță egală cu distanța lui  $P$  față de planul dat. Pentru cazul particular în care planul de simetrie este unul din planele sistemului de coordonate, simetricul unui punct este punctul care are coordonatele pe cele două axe ale planului egale și coordonata pe cealaltă axă a sistemului de coordonate cu valoare negată. Se pot deduce simplu matricele de transformare de simetrie relative la planele sistemului de referință:

$$S_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_{zx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.34)$$

Transformarea de simetrie față de un plan oarecare se poate calcula folosind o secvență de transformări elementare.

Dacă se consideră transformarea de simetrie ca o transformare aplicată unui sistem de coordonate dat, sistemul de coordonate rezultat are convenția de orientare a axelor inversă față de convenția inițială. De exemplu, sistemul drept  $Oxyz$  se transformă prin simetria  $S_{xy}$  într-un sistem de coordonate orientat după regula mâinii stângi (fig. 3.10).

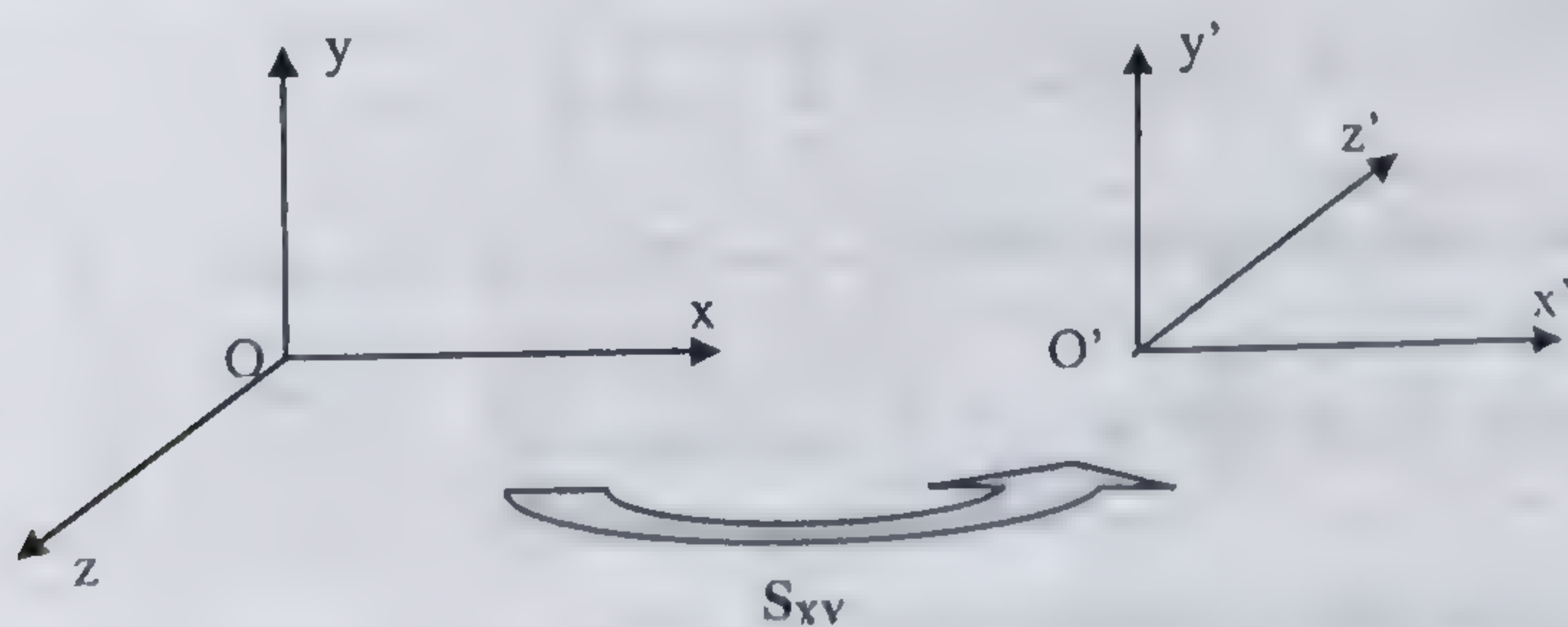


Fig. 3.10 Transformarea de simetrie  $S_{xy}$  transformă sistemul de coordonate drept  $Oxyz$  în sistemul de coordonate stâng  $O'x'y'z'$ .

**Transformarea de forfecare (shear)** este o transformare care modifică forma și dimensiunea obiectului transformat. Transformările de forfecare cele mai simple sunt cele definite față axele sistemului de coordonate.

Forfecarea după axa  $x$  a sistemului de coordonate modifică un punct  $P(x,y,z)$  în punctul  $P'(x',y',z')$ , astfel că:

$$x' = x + ay + bz; y' = y; z' = z$$



În forma matriceală în coordonate omogene, forfecarea după axa  $x$  se scrie astfel:

$$\mathbf{P}' = \mathbf{F}_x \mathbf{P}, \text{ unde } \mathbf{F}_x(a, b) = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.35)$$

În mod similar se definesc matricele de forfecare pe axele  $y$  și  $z$ . Transformarea de forfecare este folosită în definirea sistemelor de proiecție.

### 3.2 TRANSFORMĂRI GEOMETRICE ÎN PLAN

Uneori sunt necesare transformări geometrice în planul bidimensional. Exprimarea matriceală unitară a transformărilor geometrice în plan se poate face într-un sistem de coordonate omogene definit în mod asemănător sistemului de coordonate omogene în spațiu: unui punct  $P(x, y)$  în plan îi corespunde un punct  $P(X, Y, w)$ , cu relațiile între coordonate:  $x = X/w$ ;  $y = Y/w$ , unde factorul de scară  $w$  este diferit de zero.

Transformările geometrice în planul bidimensional pot fi definite prin matrice de transformare de dimensiune  $3 \times 3$ . Pentru transformările geometrice primitive în plan, matricele de transformare se deduc simplu, ca o particularizare a transformărilor geometrice spațiale. Acestea au următoarele expresii:

Translația cu valorile  $t_x$  și  $t_y$ :

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \mathbf{P}, \text{ unde } \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.36)$$

Scalarea față de origine cu factorii de scară  $s_x, s_y$ :

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \mathbf{P}, \text{ unde } \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.37)$$

Rotația față de origine cu un unghi  $\theta$ :

$$\mathbf{P}' = \mathbf{R}(\theta) \mathbf{P}, \text{ unde } \mathbf{R}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.38)$$

Transformări complexe în plan se pot defini, la fel ca și în spațiu, prin compunerea (produs de matrice) de transformări primitive. Toate transformările între sisteme de referință în plan pot fi considerate cazuri particulare ale transformărilor corespunzătoare în spațiu.



## SISTEME DE VIZUALIZARE

Un sistem de vizualizare este definit printr-o colecție de sisteme de referință, convenții de reprezentare și relații (matriceale) de transformare care permit executarea unei succesiuni de transformări având ca efect punerea în corespondență (*mapping*) a punctelor din sistemul de referință de modelare (*modelling coordinate space*) cu puncte ale suprafeței de vizualizare (*view surface*).

Într-un sistem de vizualizare, generarea imaginii scenei virtuale este un proces complex care poate fi parcurs prin reprezentări în mai multe sisteme de referință, fiecare sistem de referință facilitând specificarea și executarea anumitor operații (fig. 4.1). În multe lucrări, generarea imaginii mai este denumită și vizualizarea sau redarea scenelor (*viewing, rendering*). Pe parcursul lucrării vor fi utilizați toți acești termeni.

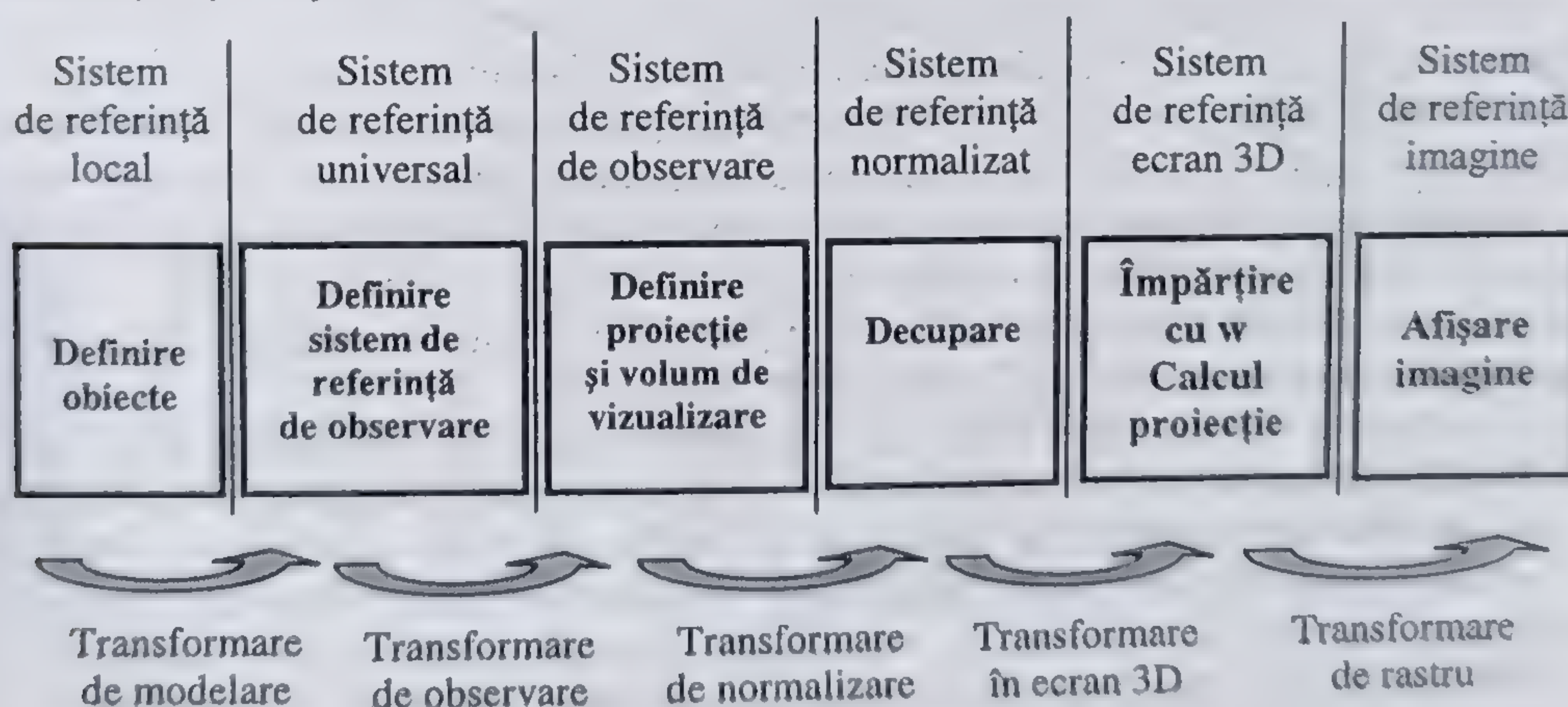


Fig. 4.1 Succesiunea operațiilor de vizualizare.

Obiectele tridimensionale componente ale scenei sunt modelate și reprezentate în sisteme de referință locale (sau de modelare), care au originea și orientarea alese cel mai convenabil pentru reprezentarea obiectelor respective.



Pentru construirea scenei, obiectele specificate în sistemul de referință de modelare sunt instanțiate prin aplicarea unei succesiuni de transformări geometrice, care constituie transformarea de modelare (descrișă în § 3.1.5.4). Prin transformarea de modelare se obține descrierea obiectelor într-un sistem de referință unic al scenei virtuale, numit sistemul de referință universal. În sistemul de referință universal mai sunt definite și alte elemente necesare redării realiste a scenelor, cum sunt sursele de lumină sau traiectoriile obiectelor mobile în scenă.

În *sistemul de referință universal* se definește un sistem de referință de observare (*view reference*), care specifică punctul și direcția din care este privită (observată) scena. Este evident că, pentru aceeași scenă, imaginea care se redă pe display depinde de punctul și direcția din care este observată scena, deci de felul în care este definit sistemul de referință de observare (care mai este denumit și sistem de referință ochi – *eye coordinate space*). Transformarea obiectelor din sistemul de referință universal în sistemul de referință de observare este numită transformare de observare (sau de vizualizare – *view transformation*; s-a evitat, însă, această traducere, datorită suprautilizării termenului de vizualizare).

În *sistemul de referință de observare* se poate face o analogie simplă cu modul în care este privită o scenă reală prin ochiul uman sau printr-un aparat fotografic (cameră). Poziția și orientarea sistemului de referință de observare corespund poziției și orientării ochiului sau camerei într-o scenă reală. Dintr-un punct de observare dat, numai o parte a scenei poate fi văzută, restul aflându-se în spatele sau în părțile laterale ale direcției de observare. Partea vizibilă din scenă (reală sau virtuală) este definită ca volum de vizualizare (*viewing volume*), care, în general, este o piramidă (sau un trunchi de piramidă) cu centrul în punctul de observare și cu direcția dată de direcția de observare. În redarea scenelor, deschiderea unghiulară a piramidei de vizualizare definește câmpul de vizualizare (*field of view*), care este o caracteristică importantă a sistemelor grafice.

Deși există și unele dispozitive de afișare tridimensională, majoritatea sistemelor grafice afișează imaginile pe o suprafață plană a unui dispozitiv de afișare (display). De aceea, în succesiunea operațiilor de vizualizare a obiectelor sau a scenelor tridimensionale, este necesară introducerea unei proiecții, care convertește spațiul tridimensional într-o reprezentare bidimensională. În general, proiecțiile transformă spațiul  $R^n$  în spațiul  $R^m$ , unde  $m < n$ .

Modul în care se realizează proiecția obiectelor tridimensionale se definește în sistemul de referință de observare prin mai mulți parametri, dintre care cei mai importanți sunt: tipul proiecției și planul de vizualizare (*view plane*). Intersecția dintre planul de vizualizare și piramida de vizualizare definește o regiune plană numită fereastră de vizualizare (*view plane window*), în care sunt reprezentate toate imaginile bidimensionale ale obiectelor (sau părților de obiecte) vizibile din scenă.

În grafica tridimensională există mai multe posibilități de definire a sistemelor de vizualizare, de la definiții simple, utile în mici programe grafice, până la definiții complete, care permit crearea simultană a mai multor imagini ale scenei, în ferestre de vizualizare diferite. În majoritatea acestor sisteme de vizualizare, din motive de eficiență și precizie a calculelor (care sunt explicate în continuare),



transformarea de proiecție se execută în două etape: prima etapă este o transformare de normalizare, care este o schimbare a sistemului de referință de la sistemul de referință de observare la sistemul de referință normalizat. Volumul de vizualizare definit în sistemul de referință de observare este transformat, în sistemul de referință normalizat, într-un volum canonic (în general, un paralelipiped dreptunghic).

În sistemul de referință normalizat (*normalized coordinates*) se execută decuparea obiectelor (*clipping*), astfel încât obiectele sau părțile din obiecte care se află în afara volumului de vizualizare (volumul canonic) sunt eliminate (ignoreate pentru operațiile următoare de redare, nu șterse din scenă).

Următoarea transformare geometrică este transformarea din sistemul normalizat în sistemul de referință ecran tridimensional (*three-dimensional scene*), prin care fereastra definită în planul de vizualizare este transformată într-o regiune corespunzătoare zonei de afișare pe display, numită poartă (*viewport*), iar coordonata  $z$  se păstrează nemodificată. În sistemul de referință ecran 3D se calculează coordonatele tridimensionale ale punctelor, din coordonatele omogene, prin împărțirea cu  $w$ , după care se execută transformarea de rastru (*rasterization*). Transformarea de rastru calculează mulțimea pixelilor care aparțin unui segment sau poligon dat prin coordonatele vârfurilor. Prin această transformare (care se mai numește și conversie de baleiere – *scan conversion*) spațiul bidimensional continuu al porții de afișare este convertit în spațiul bidimensional discret al imaginii. În general, concomitent cu transformarea de rastru sunt executate și alte operații: eliminarea suprafețelor ascunse (*hidden surface removal*), umbrirea (*shading*), texturarea (*texturing*).

Aceasta este succesiunea cea mai generală a operațiilor de vizualizare a scenelor, care este descrisă și utilizată în momentul de față în grafică și realitatea virtuală. Trebuie remarcat faptul că evoluția tehnologică, deosebit de spectaculoasă în acest domeniu, a modificat, uneori substanțial, modul de abordare a generării imaginii obiectelor tridimensionale. De exemplu, posibilitatea de implementare eficientă (uneori chiar în hardware) a operației de comparare a adâncimii punctelor a impus utilizarea algoritmului cunoscut sub numele de algoritm Z-buffer și a condus la ignorarea completă a altor algoritmi de eliminare a suprafețelor ascunse, care în sistemele mai vechi aveau o mare importanță. Se pot da numeroase exemple de modul în care schimbările tehnologice au influențat abordarea graficii tridimensionale, și, probabil, astfel de influențe vor continua să apară și în viitor. De aceea, lucrarea își propune să prezinte aspectele esențiale ale graficii utilizate în realitate virtuală, mai puțin influențate de contextul tehnologic de moment.

## 4.1 TRANSFORMAREA DE OBSERVARE

Transformarea de observare este analogă cu poziționarea unei camere fotografice sau de înregistrare, prin care se observă o anumită zonă din scena virtuală. În sistemul de referință universal sunt reprezentate toate obiectele



(modelele) scenei virtuale, iar în fiecare cadru al imaginii (*frame*), este generată imaginea acelor obiecte din scenă care sunt vizibile din punctul de observare curent (*view point*).

Transformarea de observare este, așadar, o schimbare a sistemului de coordonate de la sistemul de referință universal la sistemul de referință de observare (*view coordinate*). În majoritatea aplicațiilor de realitate virtuală (simulatoare, aplicații în arhitectură, etc) scena virtuală reprezentată în sistemul de referință universal este menținută într-o poziție constantă (cel mult existând anumite obiecte mobile în scenă), iar observatorul este cel care își modifică poziția și orientarea, și în funcție de acestea se redă imaginea corespunzătoare pe ecran. Direcția de observare a scenei este, în majoritatea sistemelor de vizualizare, direcția axei  $z$  a sistemului de referință de observare.

Localizarea și orientarea sistemului de referință de observare  $O_V x_V y_V z_V$  în raport cu sistemul de referință universal  $Oxyz$  poate fi descrisă printr-o matrice de transformare  $M$  de forma generală:

$$M = T_V R_V = \begin{bmatrix} 1 & 0 & 0 & x_V \\ 0 & 1 & 0 & y_V \\ 0 & 0 & 1 & z_V \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & c_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} & c_{31} & x_V \\ c_{12} & c_{22} & c_{32} & y_V \\ c_{13} & c_{23} & c_{33} & z_V \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

Matricea componentă  $T_V$  este matricea de translație a sistemului de referință de observare în raport cu sistemul de referință universal:  $x_V, y_V, z_V$  sunt coordonatele punctului de observare  $O_V$  în sistemul  $Oxyz$ . Matricea de rotație  $R_V$  reprezintă orientarea sistemului de referință de observare față de sistemul de referință universal: coeficienții  $c_{11}, c_{12}, c_{13}$  sunt cosinușii directori ai axei  $O_V x_V$  față de axele  $Ox, Oy$ , respectiv  $Oz$ ; coeficienții  $c_{21}, c_{22}, c_{23}$  sunt cosinușii directori ai axei  $O_V y_V$  față de axele  $Ox, Oy$ , respectiv  $Oz$ ; coeficienții  $c_{31}, c_{32}, c_{33}$  sunt cosinușii directori ai axei  $O_V z_V$  față de axele  $Ox, Oy$ , respectiv  $Oz$ .

Fiind dată matricea de poziționare și orientare a sistemului de referință observator relativ la sistemul de referință universal  $M$ , un punct oarecare  $P(x, y, z)$  din sistemul  $Oxyz$  se transformă în punctul  $P_V(x_V, y_V, z_V)$  în sistemul de referință observator prin înmulțire cu matricea inversă, deci:

$$P_V = M^{-1}P = R_V^{-1}T_V^{-1}P = M_V P \quad (4.2)$$

unde matricea de transformare de observare  $M_V$  este inversa matricei de localizare a sistemului de referință de observare relativ la sistemul de referință universal, adică  $M_V = M^{-1}$ .

Matricea de rotație mai poate fi precizată și printr-o succesiune de rotații în raport cu axele sistemului de referință universal. Cea mai obișnuită convenție pentru ordinea de specificare a rotațiilor este: ruliu cu unghiul  $\rho$  (după axa  $z$ ), tangaj cu unghiul  $\theta$  (după axa  $x$ ) și girație cu unghiul  $\chi$  (față de axa  $y$ ). În această situație:

$$R_V = R_Y(\chi)R_X(\theta)R_Z(\rho) \quad (4.3)$$



În aceste relații de transformare de observare s-a presupus că sistemul de referință de observare este tot un sistem drept, la fel ca și sistemul de referință universal. Există însă și sisteme de vizualizare în care se definește sistemul de referință de observare ca un sistem orientat după regula mâinii stângi. Pentru schimbarea sistemului de referință de observare drept în sistem de referință stâng, se adaugă o transformare de oglindire (simetrie), de exemplu cu matricea  $S_{XY}$ , iar matricea de transformare de observare devine  $M'_V$ , unde:

$$M'_V = S_{XY} M_V$$

Dat fiind că nu există o predilecție certă pentru una din aceste convenții posibile (sistem de referință de observare drept sau stâng), este necesar să fie precizată de fiecare dată convenția folosită, ceea ce se va face și pe parcursul acestei lucrări.

În deducerea relațiilor (4.2) și (4.3) s-a presupus că este cunoscută matricea  $M$  de localizare a sistemului de referință de observare față de sistemul de referință universal și atunci matricea de transformare de observare  $M_V$  se calculează ca matrice inversă a acesteia. Există situații în care matricea  $M$  nu este cunoscută și atunci matricea de transformare de observare  $M_V$  (sau  $M'_V$ ) trebuie să fie calculată prin diferite transformări geometrice și raționamente. În exemplul următor se urmărește un raționament de calcul al transformării de observare, pentru o anumită cerință de observare a scenei.

#### ■ Exemplul 4.1

Se poate defini un sistem de referință de observare pornind de la alte condiții decât cele prezentate anterior. De exemplu, se cere definirea matricei de transformare de observare în situația în care obiectul care trebuie să fie văzut se află în centrul sistemului de referință universal, iar punctul de observare este un punct dat în spațiul tridimensional,  $E(x_E, y_E, z_E)$ .

Pentru ca obiectul aflat în centrul sistemului de referință universal să fie văzut din punctul  $E$ , se definește sistemul de referință de observare drept cu centrul în punctul  $E$ , și cu axa  $z_V$  orientată pe dreapta  $OE$  (cu sensul de la  $O$  către  $E$ ). Direcția de observare într-un sistem de referință drept este definită (în majoritatea sistemelor de vizualizare) în sens invers direcției axei  $z$ , deci către centrul  $O$ .

Calculele de transformare se efectuează mai ușor dacă se dau coordonatele sferice ale punctului  $E$ : distanța  $\mu$  a punctului  $E$  față de origine și unghiurile  $\theta$  și  $\phi$  (fig. 4.2). Coordonatele carteziane ale punctului  $E$  pot fi calculate în funcție de coordonatele sferice:

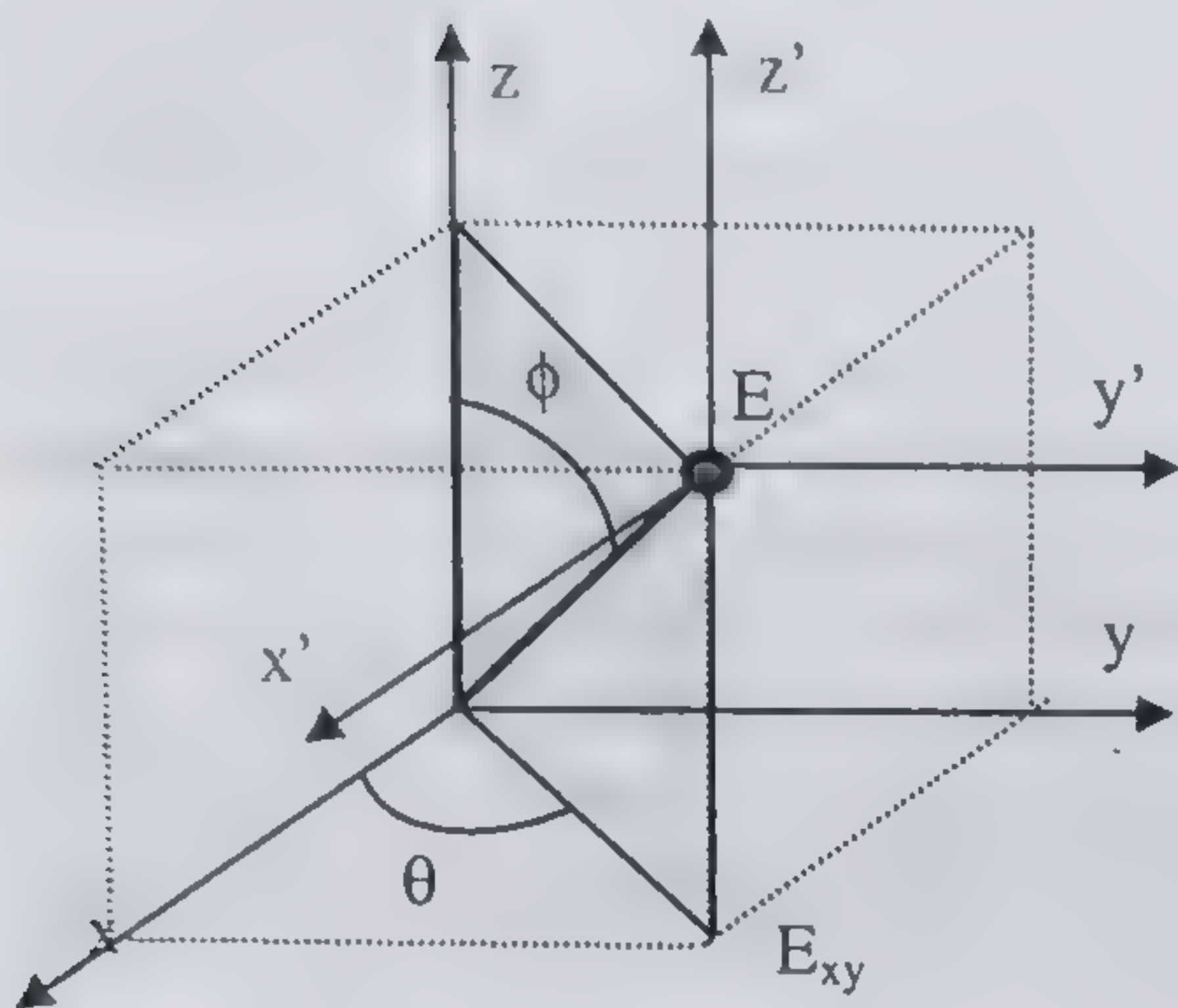
$$\begin{cases} x_E = \mu \sin \phi \cos \theta \\ y_E = \mu \sin \phi \sin \theta \\ z_E = \mu \cos \phi \end{cases}$$

Pentru transformarea unui obiect din sistemul de referință universal în sistemul de referință de observare astfel definit, sunt necesare trei transformări elementare. În fig. 4.2 sunt reprezentate primele două transformări componente ale transformării de observare.

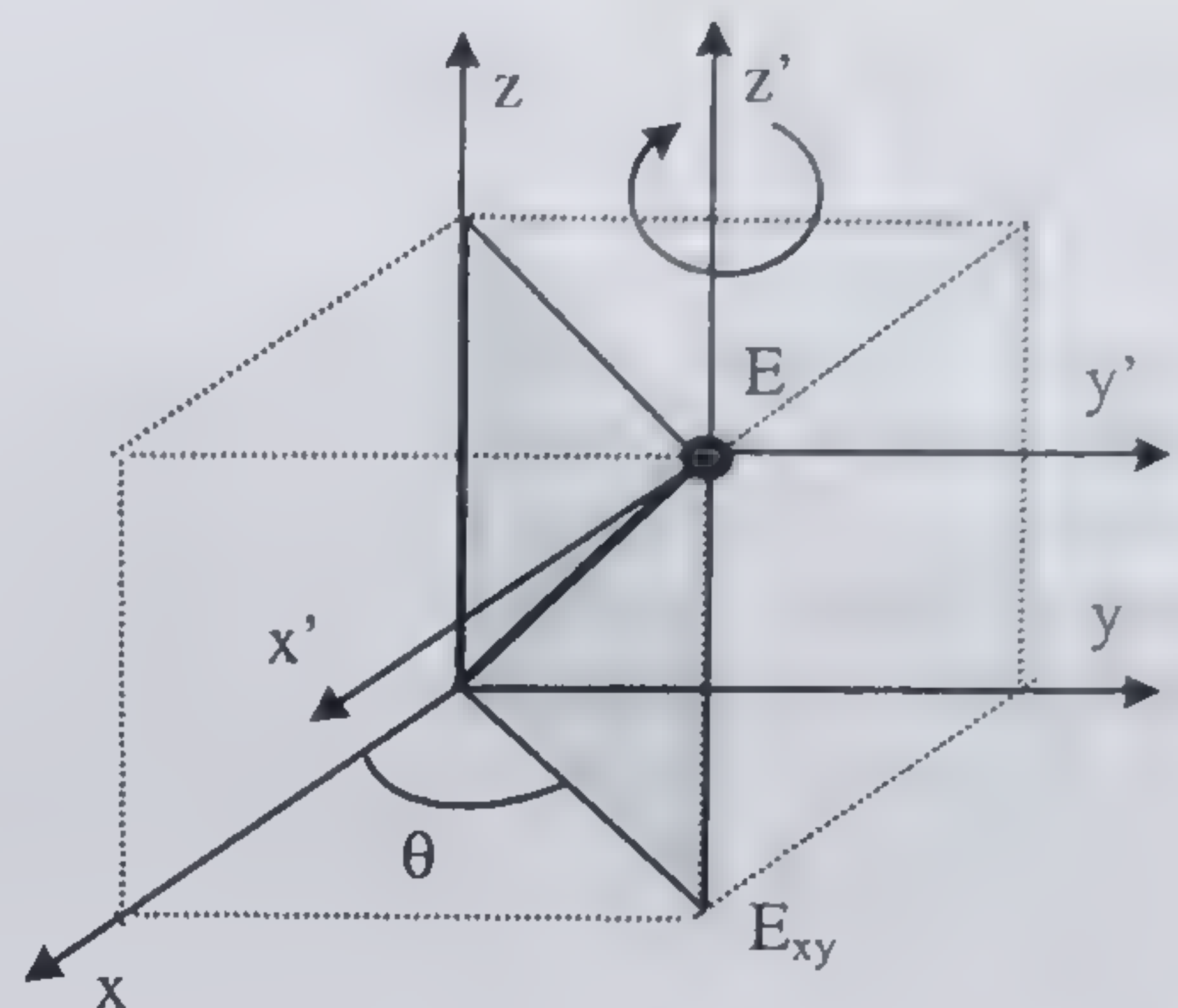


Translația  $T_1 = T(-x_E, -y_E, -z_E)$  transformă sistemul de referință universal într-un sistem de referință cu centrul în punctul E și cu axe  $x'$ ,  $y'$ ,  $z'$  paralele cu axele sistemului de referință universal.

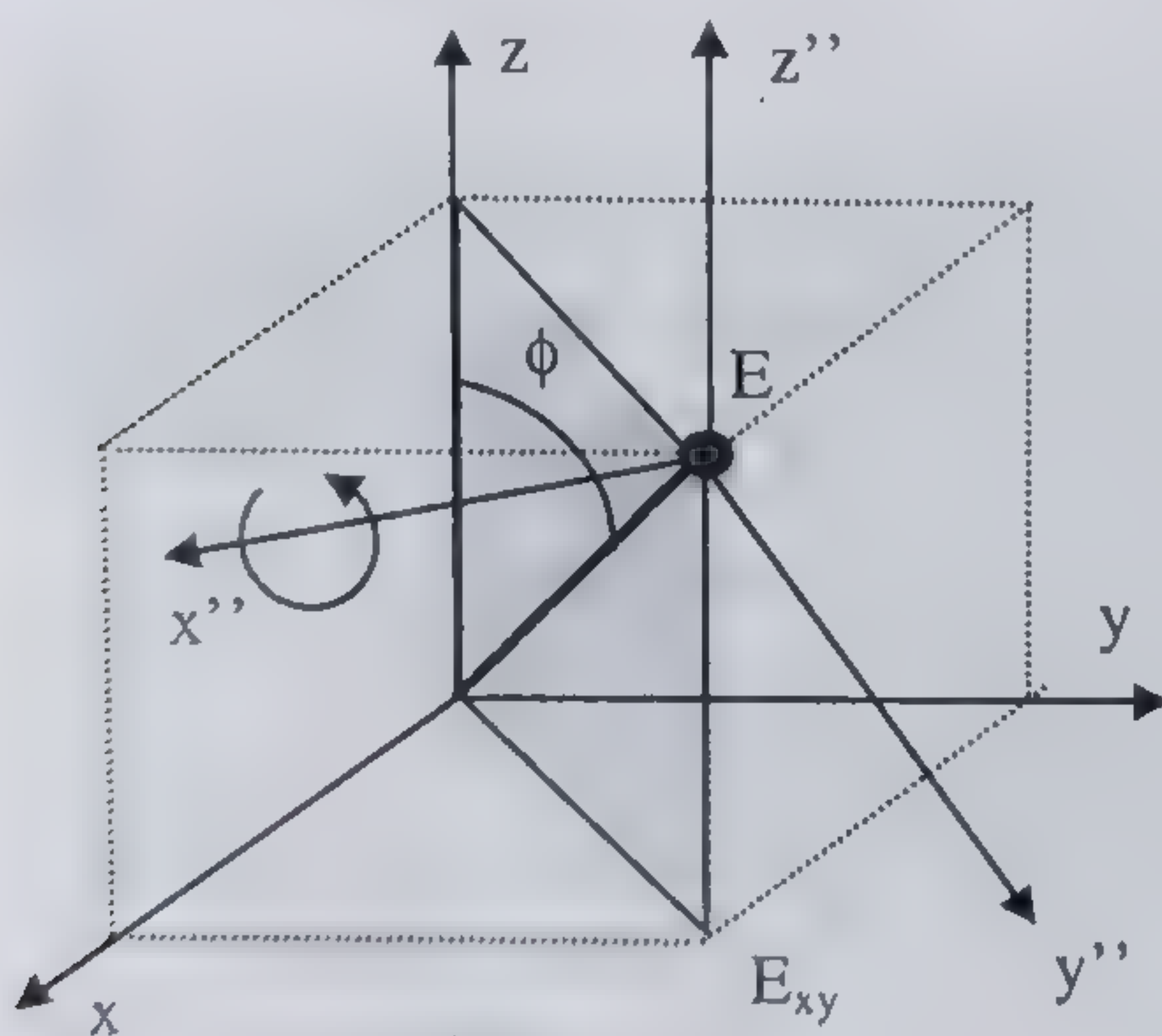
$$T_1 = \begin{bmatrix} 1 & 0 & 0 & -\mu \cos\theta \sin\phi \\ 0 & 1 & 0 & -\mu \sin\theta \sin\phi \\ 0 & 0 & 1 & -\mu \cos\phi \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



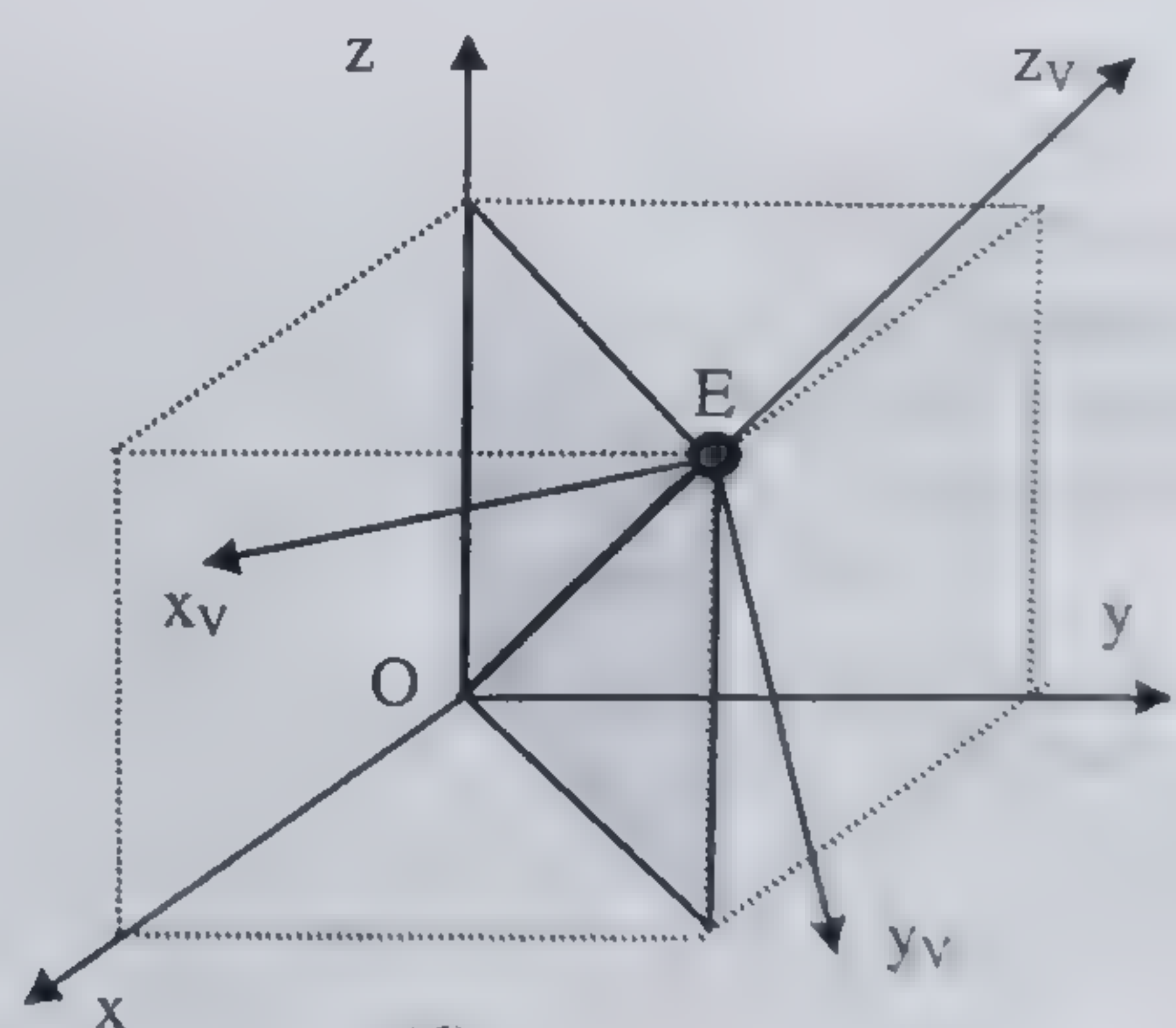
(a)



(b)



(c)



(d)

Fig. 4.2 Transformarea de observare: (a) translația (b) rotația în raport cu axa  $z'$  (c) rotația în raport cu axa  $x''$  (d) sistemul de referință rezultat.

Orientarea sistemului de referință de observare astfel încât axa  $z'$  să se suprapună peste dreapta OE se obține prin două rotații.

Prima rotație se efectuează în raport cu axa  $z'$  cu un unghi de  $90^\circ - \theta$  în sens negativ, astfel încât noua axă  $x''$  obținută să fie normală la planul care trece prin axa  $z$  și punctul E (planul  $OEE_{xy}$ ) (fig. 4.2(b)). Această operație este echivalentă cu o rotație în sens direct aplicată asupra punctelor, deci cu matricea  $R_2 = R_z(90^\circ - \theta)$ , unde:



$$\mathbf{R}_2 = \begin{bmatrix} \sin \theta & -\cos \theta & 0 & 0 \\ \cos \theta & \sin \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

După această rotație, sistemul de referință de observare are axa  $x''$  perpendiculară pe planul  $OEE_{xy}$  și axa  $Ez''$  în planul  $OEE_{xy}$ .

A doua rotație este o rotație a sistemului de coordonate  $Ex''y''z''$  în sens negativ cu un unghi egal cu  $\phi$  în raport cu axa  $Ex''$ , pentru a suprapune axa  $Ez''$  peste dreapta  $OE$  (fig. 4.2(a)). Această rotație este echivalentă cu o rotație inversă aplicată obiectelor, deci cu matricea  $\mathbf{R}_3 = \mathbf{R}_X(\phi)$ :

$$\mathbf{R}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rezultă matricea de transformare de observare:

$$\mathbf{M}_V = \mathbf{R}_X(\phi) \mathbf{R}_Z(90^\circ - \theta) \mathbf{T}(-x_E, -y_E, -z_E)$$

$$\mathbf{M}_V = \begin{bmatrix} \sin \theta & -\cos \theta & 0 & 0 \\ \cos \theta \cos \phi & \sin \theta \cos \phi & -\sin \phi & 0 \\ \cos \theta \sin \phi & \sin \theta \sin \phi & \cos \phi & -\mu \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformarea unui punct  $P(x, y, z)$  din sistemul de referință universal în punctul  $P(x_V, y_V, z_V)$  în sistemul de referință de observare se obține astfel:

$$[x_V \ y_V \ z_V \ 1]^T = \mathbf{M}_V [x \ y \ z \ 1]^T$$

În fig. 4.3 este reprezentată imaginea unui cub cu centrul în origine și laturile egale cu 2 și paralele cu axele de coordonate, observat din punctul  $E(10, 10, 10)$ . Pentru punctul  $E$ :

$$\mu = 10\sqrt{3}$$

$$\cos \theta = \sqrt{2}/2, \sin \theta = \sqrt{2}/2, \theta = 45^\circ$$

$$\cos \phi = \sqrt{3}/3, \sin \phi = \sqrt{6}/3, \phi = 54^\circ 45'$$

Matricea de transformare de observare are valoarea:

$$\mathbf{M}_V = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{6}/6 & \sqrt{6}/6 & -\sqrt{6}/3 & 0 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 & -10\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



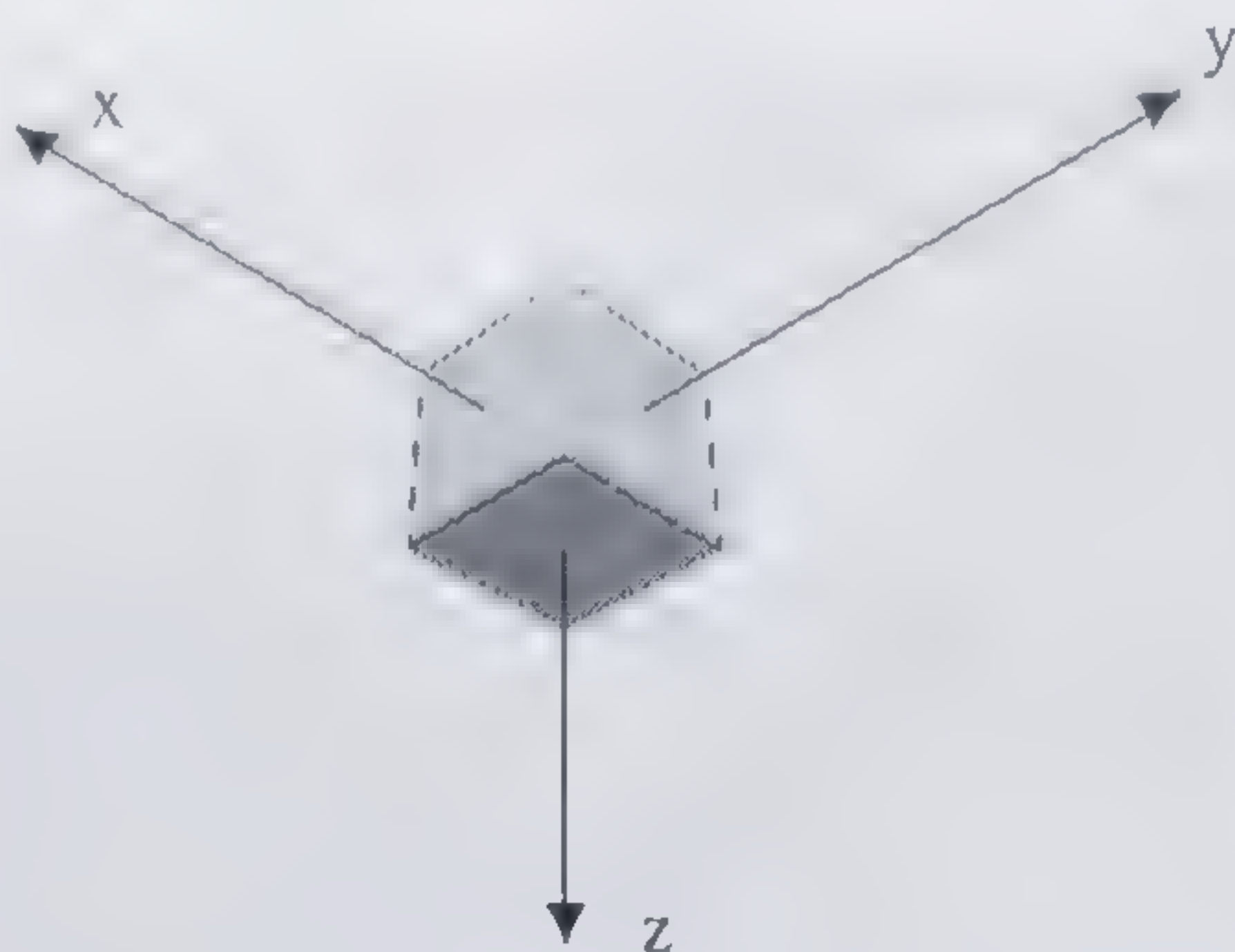


Fig. 4.3 Exemplu de observare către centrul sistemului de referință universal.

Centrul  $O$  al sistemului de referință universal se transformă în punctul  $O'(0,0,-9\sqrt{3})$  în sistemul de referință de observare, iar vârful cubului  $P(1,1,1)$ , se transformă în punctul  $P'(0,0,-10\sqrt{3})$ . Direcția de observare trece printr-un vârf al cubului și prin centrul lui.

## 4.2 TRANSFORMAREA DE PROIECȚIE

Proiecția unui obiect tridimensional pe un plan (numit plan de proiecție sau plan de vizualizare – *view plane*) este formată din mulțimea punctelor de intersecție între acest plan și dreptele numite proiectori, care pornesc dintr-un punct fix numit centru de proiecție și trec prin fiecare punct al obiectului.

În funcție de distanța centrului de proiecție față de planul de proiecție, proiecțiile plane se clasifică în proiecții perspectivă și proiecții paralele. În proiecțiile perspectivă centrul de proiecție se află la distanță finită față de planul de vizualizare, iar în proiecțiile paralele centrul de proiecție este la infinit. În cazul proiecțiilor paralele, proiectorii sunt drepte paralele și au o direcție comună, numită *direcție de proiecție*.

În grafica tridimensională se folosește cel mai frecvent proiecția perspectivă, deoarece această proiecție redă cel mai bine efectul de descreștere a dimensiunilor obiectelor odată cu creșterea distanței acestora față de observator.

Modul de definire a punctului de observare a scenei, a centrului de proiecție și a planului de vizualizare depind de sistemul de vizualizare folosit. Un studiu atent al sistemelor de vizualizare este necesar din mai multe motive. Chiar dacă un utilizator nu trebuie să proiecteze el însuși sistemul de vizualizare, exploatarea oricărui sistem necesită o înțelegere precisă a modului în care diferiți parametri determină imaginea obținută pe display. Combinațiile posibile sunt în număr destul de mare, astfel încât șansa de a alege diferiți parametri ai unui sistem de vizualizare, fără a fi cunoscute condițiile în care lucrează acesta, este foarte redusă.



### 4.2.1 PROIECTIA PARALELĂ

În proiecția paralelă, centrul de proiecție este la infinit, iar toate dreptele de proiecție sunt paralele între ele, având o direcție numită direcția de proiecție.

Proiecțiile paralele sunt foarte rar folosite în grafica destinată sistemelor de realitate virtuală, dat fiind că imaginea rezultată nu dă informații asupra distanței obiectelor în scenă.

Planul de proiecție al unei proiecții paralele poate fi perpendicular pe direcția de proiecție, și atunci proiecția se numește *proiecție ortografică*, sau poate avea o înclinare oarecare față de direcția de proiecție, și atunci proiecția se numește *proiecție oblică*.

Dintre proiecțiile ortografice, cele mai utilizate sunt proiecțiile ortografice în care planul de proiecție este unul din planurile sistemului de coordonate. Proiecțiile ortografice sunt folosite frecvent în desenul tehnic și, de asemenea, ca o etapă intermediară în definirea sistemelor de vizualizare normalizate, care vor fi prezentate în paragraful următor. Matricea de transformare de proiecție ortografică în planul  $z = 0$  are următoarea expresie:

$$P_Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

Prin proiecție ortografică în planul  $z = 0$ , un punct  $P(x, y, z)$  se transformă în punctul  $P'(x, y, 0)$ , care are aceleași coordonate  $x$  și  $y$ , iar  $z$  are valoarea zero.

În mod similar, se definesc matricele de proiecție ortografică în planele  $x = 0$  și  $y = 0$ :

$$P_X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad P_Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

### 4.2.2 PROIECTIA PERSPECTIVĂ

Există mai multe modalități de definire a unei proiecții perspective, de la o definire minimală, prin care se poate realiza proiecția punctelor din spațiu pe un plan, până la definirea adoptată în sistemele de vizualizare complexe, în care se specifică volumul de vizualizare, centrul de proiecție și alți parametri. În continuare, vor fi prezentate câteva sisteme de vizualizare cu proiecție perspectivă, începând cu cel mai simplu sistem și până la sistemul complet de vizualizare care se conformează standardelor GKS și PHIGS.

Proiecția perspectivă se poate defini mai intuitiv pornind de la sistemul de observare considerat ca un sistem de referință stâng. Centrul de proiecție se fixează în centrul sistemului de referință de observare  $O_v$ , direcția de observare (de privire)



este în sensul pozitiv al axei  $O_V z_V$ , iar planul de vizualizare  $\pi$  este un plan perpendicular pe axa  $O_V z_V$ , la distanța  $d$  față de centru (fig. 4.4(a)).

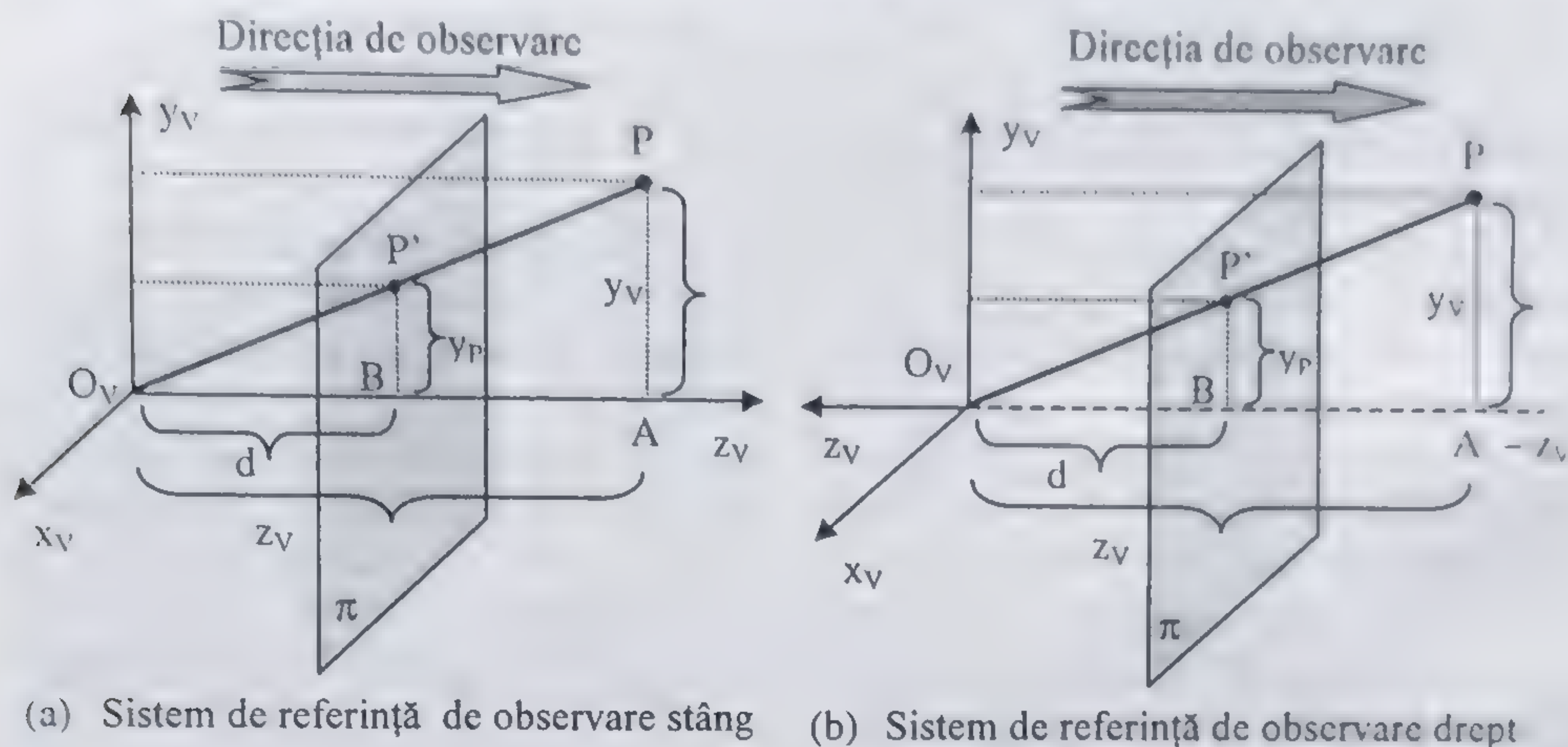


Fig. 4.4 Proiecția perspectivă definită prin centrul de proiecție  $O_V$  și planul de proiecție  $\pi$  perpendicular pe axa  $O_V z_V$ .

Punctul  $P(x_V, y_V, z_V)$  în sistemul de referință de observare se proiectează în punctul  $P'(x_P, y_P, z_P)$ , aflat la intersecția dreptei  $O_V P$  cu planul de vizualizare. Din triunghiurile asemenea  $O_V P' A$  și  $O_V P B$ , rezultă:

$$y_P = \frac{d y_V}{z_V}$$

În mod asemănător se poate deduce valoarea lui  $x_P$ :

$$x_P = \frac{d x_V}{z_V} \quad (4.6)$$

Aceste relații se pot exprima matriceal prin definirea matricei de proiecție perspectivă  $M_P$  astfel:

$$M_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \quad (4.7)$$

Transformarea de proiecție se poate scrie atunci:

$$P' = M_P P, \quad \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ w_P \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} X_V \\ Y_V \\ Z_V \\ w_V \end{bmatrix}$$



Rezultă:

$$\begin{cases} X_p = X_v \\ Y_p = Y_v \\ Z_p = Z_v \\ w_p = Z_v / d \end{cases} \quad \text{și} \quad \begin{cases} x_p = X_p / w_p = d X_v / Z_v = d x_v / z_v \\ y_p = Y_p / w_p = d Y_v / Z_v = d y_v / z_v \\ z_p = Z_p / w_p = d \end{cases} \quad (4.8)$$

Se observă că expresiile lui  $x_p$ ,  $y_p$  și  $z_p$  obținute prin calcul matriceal sunt identice cu cele din relațiile (4.6), rezultate din construcția geometrică prezentată în fig. 4.4(a). Proiecția perspectivă necesită deci o împărțire a coordonatelor  $x$  și  $y$  cu distanța  $z$ , și, într-adevăr, în proiecția perspectivă imaginea unui obiect este cu atât mai mică cu cât se află la distanță mai mare de punctul de observare.

În acest sistem de vizualizare simplu s-a definit sistemul de referință de observare ca un sistem stâng. Se poate defini acest sistem ca un sistem de referință drept, iar direcția de observare îndreptată spre  $-z_v$ . Planul de vizualizare este poziționat perpendicular pe axa  $z$ , la distanță egală cu  $d$  față de origine (fig. 4.4(b)),  $d$  având valoare negativă. Urmând un raționament asemănător cu cel de mai înainte, rezultă matricea de proiecție perspectivă cu aceeași expresie (4.7), în care  $d$  are o valoare negativă. Coordonatele punctului transformat sunt date de relațiile (4.8).

Se remarcă echivalența rezultatelor obținute, iar convenția de sistem de referință de observare stâng este utilă doar pentru a se urmări mai ușor raționamentul de calcul.

### 4.2.3 SISTEMUL DE REFERINȚĂ NORMALIZAT

Sistemul de proiecție simplu definit anterior, în care toate obiectele din scenă sunt proiectate pe planul de vizualizare, are mai multe inconveniente și nu poate fi folosit decât în aplicații foarte simple, în care se proiectează obiecte aflate tot timpul în câmpul de observare. Nu aceasta este situația imaginilor generate în realitatea virtuală în care se explorează o scenă virtuală de dimensiuni mari, din care numai o parte este vizibilă și deci redată pe ecran.

Transformarea de proiecție perspectivă este o transformare costisitoare, datorită faptului că necesită operații de împărțire, care consumă timp de calcul ridicat, dacă este executată prin program, sau circuite complexe, dacă este implementată hardware. Execuția ei pentru toate obiectele scenei, inclusiv pentru cele care nu sunt vizibile, deci nu sunt reprezentate pe display și nu necesită coordonatele în planul de vizualizare, este costisitoare și inutilă. În plus, operația de decupare (*clipping*) a părților din obiectele tridimensionale care nu sunt vizibile dintr-un anumit punct de observare nu se poate executa eficient după proiecție și este necesară execuția ei înainte de proiectarea coordonatelor în planul de vizualizare. Asupra acestui aspect se va reveni în § 4.5.2.

Este evident că, pentru redarea eficientă a scenelor tridimensionale, este necesară definirea unui volum de vizualizare (*view volume*), cunoscut sub numele de piramida de vizualizare sau trunchiul de piramidă de vizualizare (*frustum*).



Trunchiul de piramidă de vizualizare se definește în sistemul de referință de observare. În forma cea mai simplă, trunchiul de piramidă de vizualizare este mărginit de patru plane laterale care trec prin punctul de observare  $O_v$  și de două plane perpendiculare pe axa  $O_v z_v$ , planul apropiat (*near*) pe care se execută proiecția, și planul depărtat (*far*). În fig. 4.5 este prezentat volumul de vizualizare într-un sistem de referință de observare stâng și direcția de observare către  $z_v$  pozitiv.

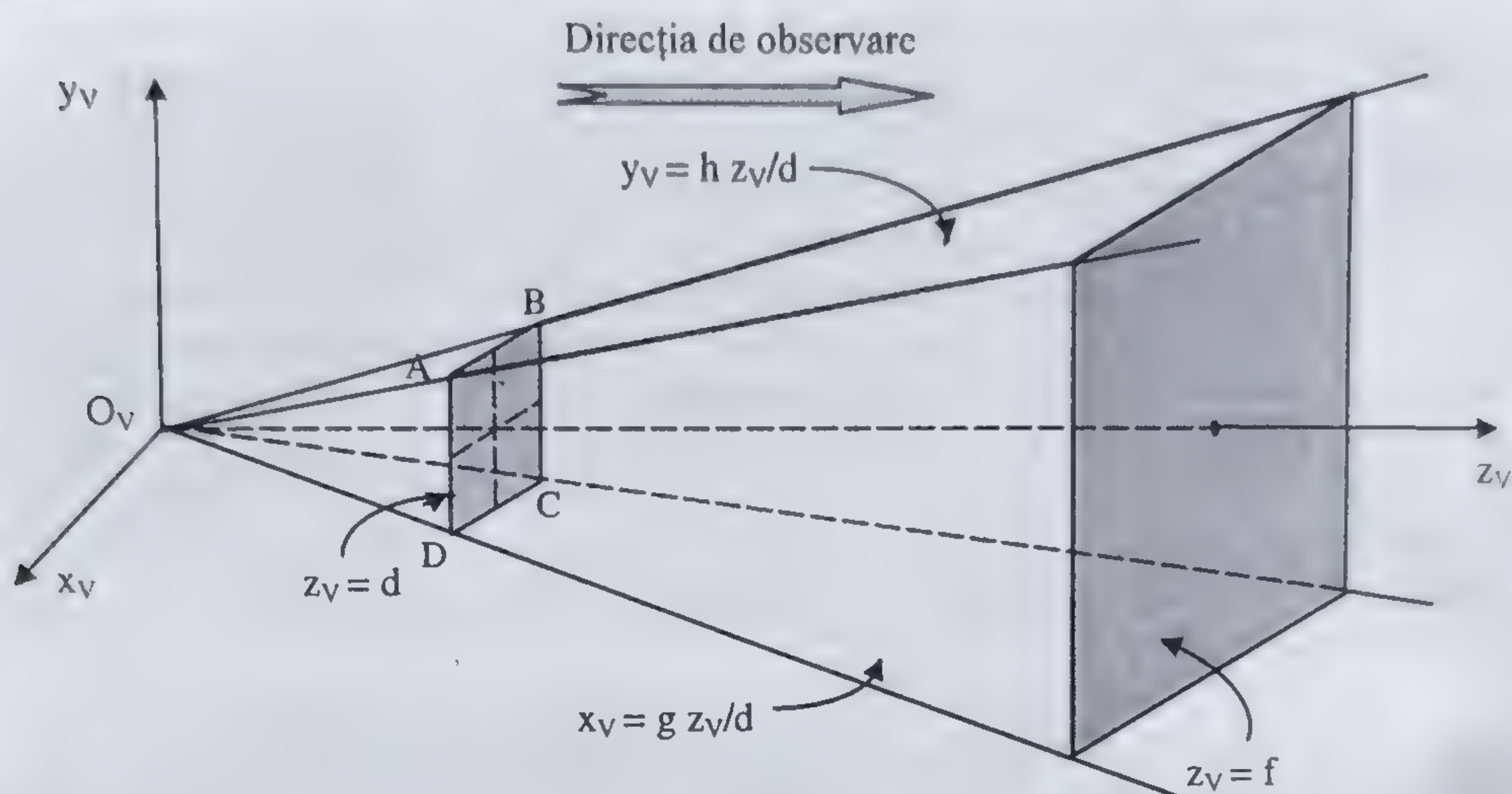


Fig. 4.5 Definirea trunchiului de piramidă de vizualizare (frustum) în sistemul de referință de observare stâng.

Intersecția dintre cele patru plane laterale și planul de vizualizare (care, în această definiție, este și planul apropiat de vizibilitate) determină o suprafață dreptunghiulară ABCD în planul de vizualizare în care vor fi proiectate toate obiectele vizibile ale scenei. Această suprafață este fereastra de vizualizare (*view plane window*). Ecuațiile celor șase plane ale trunchiului de piramidă regulată sunt:

$$\begin{cases} x_v = \pm g z_v / d \\ y_v = \pm h z_v / d \\ z_v = d \\ z_v = f \end{cases} \quad (4.9)$$

unde:  $2g$  – dimensiunea laturii orizontale a ferestrei de vizualizare (latura AB)  
 $2h$  – dimensiunea laturii verticale a ferestrei de vizualizare (latura BC)  
 $d$  – distanța planului de vizualizare și a planului apropiat  
 $f$  – distanța planului depărtat.

Semnificația volumului de vizualizare este evidentă: dintre toate obiectele din scena virtuală sunt vizibile și redată în fereastra de vizualizare numai acele obiecte sau părți din obiecte care sunt cuprinse în acest volum. Trunchiul de



piramidă de vizualizare definește în acest fel și volumul de decupare, operație care se execută într-unul din sistemele de referință care se definesc în cadrul transformării de proiecție, și anume sistemul de referință normalizat. Acest sistem de referință se mai numește și sistem de referință canonic. Motivele pentru care decuparea se efectuează cel mai eficient în sistemul de referință normalizat vor fi explicate ulterior.

Sistemul de referință normalizat este un sistem de coordonate tridimensional în care trunchiul de piramidă de vizualizare se transformă într-un volum canonic, care, în general, este un paralelipiped dreptunghic. Pentru sistemul de proiecție definit în acest paragraf, *volumul canonic* este un paralelipiped dreptunghic cu latura bazei egală cu 2 și înălțimea 1 (după axa  $z_N$ ).

Introducerea acestui nou sistem de referință permite abordarea transformării de proiecție perspectivă în două etape: o transformare din sistemul de referință de observare în sistemul de referință normalizat (numită transformare de normalizare), urmată de calculul proiecției prin împărțirea cu  $w$  a coordonatelor omogene. Între cele două etape ale transformării de proiecție perspectivă se execută o operație importantă în grafica tridimensională, decuparea obiectelor relativ la volumul de vizualizare normalizat (volumul canonic).

Pentru transformarea de normalizare (transformarea de la sistemul de referință de observare la sistemul de referință normalizat), se impun următoarele cerințe:

- (1) Se normalizează coordonata  $z_N$ , care reprezintă adâncimea, astfel încât să se obțină precizie maximă în operațiile de comparație a adâncimii punctelor.
- (2) Liniile din sistemul de referință observator trebuie să se transforme în linii în sistemul de referință normalizat.
- (3) Suprafețele plane din sistemul de referință observator trebuie să se transforme în suprafețe plane în sistemul de referință normalizat.

S-a demonstrat [New81] că aceste cerințe sunt satisfăcute de transformarea:

$$z_N = a + b/z_V$$

unde  $a$  și  $b$  sunt constante. Aceste constante se determină din următoarele condiții:

- Se alege  $b < 0$ , astfel ca  $z_N$  să crească atunci când crește  $z_V$ . Această condiție conservă noțiunea intuitivă de adâncime: un punct are coordonata  $z_N$  cu atât mai mare cu cât este mai departe de observator.
- Se normalizează domeniul de variație al lui  $z_N$ , astfel ca domeniul  $z_V \in [d, f]$  să se transforme în domeniul  $z_N \in [0, 1]$ .

Rezultă următoarele relații care definesc transformarea de normalizare:

$$x_N = \frac{d}{g} \frac{x_V}{z_V}; \quad y_N = \frac{d}{h} \frac{y_V}{z_V}; \quad z_N = \frac{f}{f-d} \left( 1 - \frac{d}{z_V} \right) \quad (4.10)$$



Coordonatele omogene ale unui punct în sistemul normalizat au expresiile:

$$\begin{cases} X_N = d x_V / g \\ Y_N = d y_V / h \\ Z_N = z_V f / (f - d) - df / (f - d) \\ w_N = z_V \end{cases} \quad (4.11)$$

Din aceste expresii se deduce matricea de transformare de normalizare  $M_N$  care efectuează transformarea de la sistemul de referință observator la sistemul de referință normalizat:

$$M_N = \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix} = M_N \begin{bmatrix} x_V \\ y_V \\ z_V \\ 1 \end{bmatrix} \quad (4.12)$$

Matricea de transformare de normalizare  $M_N$  poate fi exprimată ca un produs de două matrice astfel:

$$M_N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = M_{N2} M_{N1}$$

Această reprezentare permite analiza mai detaliată a transformării de normalizare. Sistemul de referință normalizat astfel definit este un sistem de referință stâng. Matricea  $M_{N1}$  este o matrice de scalare neuniformă prin care trunchiul de piramidă de vizualizare este transformat într-un trunchi de piramidă regulată cu unghiul la vârful piramidei de  $90^\circ$ . Matricea  $M_{N2}$  transformă acest trunchi de piramidă regulată în volumul canonic, care este un paralelipiped dreptunghic (fig. 4.6).

În sistemul de referință normalizat se execută operația de decupare a obiectelor în coordonate normalizate omogene. În urma acestei operații, din totalitatea obiectelor scenei virtuale sunt reținute numai acele obiecte sau părți (rezultate prin decupare) aflate în interiorul volumului de vizualizare canonic, ceea ce înseamnă că, înainte de transformarea de normalizare, acestea se aflau în interiorul trunchiului de piramidă de vizualizare.

Din coordonatele omogene  $X_N, Y_N, Z_N, w_N$  se pot deduce coordonatele  $x_N, y_N$  și  $z_N$  ale punctului în sistemul de referință normalizat tridimensional:

$$x_N = X_N / w_N = \frac{d x_V}{g z_V}; \quad y_N = Y_N / w_N = \frac{d y_V}{h z_V}; \quad z_N = Z_N / w_N = \frac{f}{f-d} \left( 1 - \frac{d}{z_V} \right)$$

S-au obținut, evident, expresii identice cu cele din relațiile (4.10), de la care s-a pornit pentru definirea transformării de proiecție perspectivă.



Coordonatele  $x_N$  și  $y_N$  în sistemul de referință normalizat reprezintă coordonatele proiecției ortografice în planul  $z_N = 0$  (care este planul de vizualizare) a punctului corespunzător. Este de remarcat faptul că transformarea de proiecție perspectivă din sistemul de referință de observare a devenit o transformare de proiecție ortografică în sistemul de referință normalizat. Dar aceste operații de trecere de la coordonate omogene la coordonate tridimensionale și proiecția ortografică nu se efectuează în sistemul normalizat ci într-un alt sistem de referință, sistemul ecran 3D, obținut prin transformarea ferestrei de vizualizare în poartă de afișare.

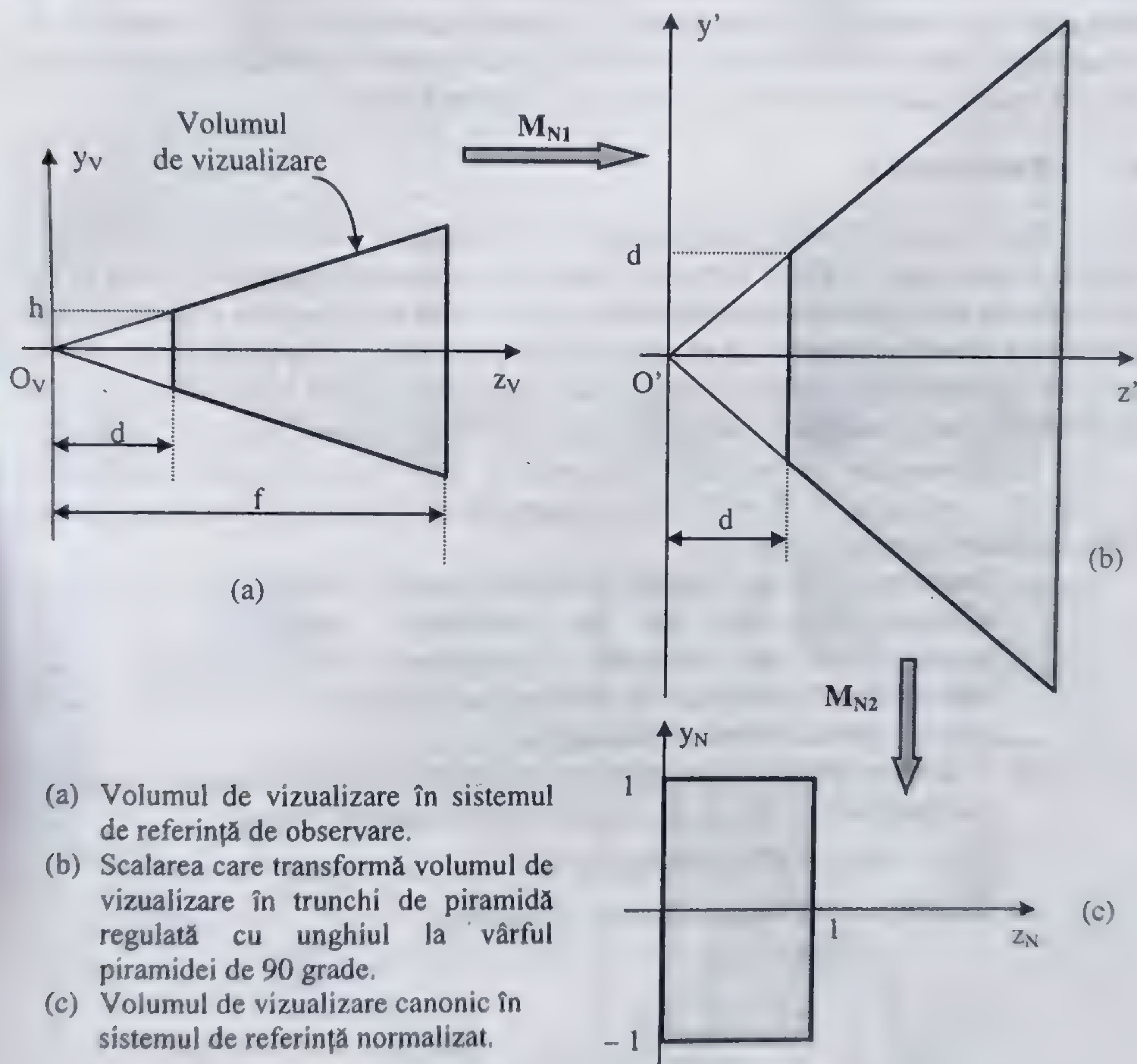


Fig. 4.6 Etapele transformării de normalizare.

Unghiul dintre planele laterale ale piramidei de vizualizare (care trec prin axa  $O_v y_v$  și au ecuațiile  $x_v = \pm g z_v/d$ ) se numește unghi de vizualizare pe orizontală (*horizontal field of view*—*fovx*). Unghiul dintre planele piramidei de vizualizare care trec prin axa  $O_v x_v$  se numește unghi de vizualizare pe verticală



(vertical field of view – fovy). Valorile acestora se deduc din parametrii de proiecție perspectivă:

$$\text{fovx} = 2 \arctan \frac{g}{d}, \quad \text{fovy} = 2 \arctan \frac{h}{d} \quad (4.13)$$

Valorile unghiurilor de vizualizare variază pentru diferite sisteme grafice în funcție de dispozitivele de afișare folosite.

O altă convenție de definire a unui sistem de vizualizare este aceea de a considera sistemul de referință de observare ca un sistem drept, iar direcția de observare și volumul de vizualizare îndreptate către  $z_v$  negativ. Urmărind un raționament asemănător celui anterior, se obține aceeași expresie a matricei de normalizare (relația 4.12), în care valorile  $d$ , și  $f$  au valori negative, iar sistemul de referință normalizat rezultat este un sistem de referință stâng.

#### ■ Exemplul 4.2

Se reia scena virtuală din exemplul 3.2. Imaginile din fig.4.7 corespund obiectelor instanțiate în scena virtuală. Axele de coordonate reprezentate sunt axele sistemului de referință universal. Sistemul de referință de observare este un sistem de referință drept, iar direcția de observare îndreptată către  $z_v$  negativ. Originea  $O_v$  are valori diferite în diferitele imagini ale figurii, așa cum este indicat pentru fiecare dintre ele, și orientare dată prin unghiurile de rotație, tangaj, ruliu ( $\chi, \theta, \rho$ ).

Transformarea de proiecție perspectivă este definită prin parametrii:  $d = -1$ ,  $f = -40$ ,  $g = 1$ ,  $h = 1$ . Succesiunea de transformări efectuate pentru obținerea acestor imagini este:

- Transformarea de modelare, diferită pentru fiecare din cuburi, cu matricele  $M_1, M_2, M_3, M_4$  calculate în exemplul 3.3. După transformarea de modelare (instanțiere), toate obiectele sunt reprezentate în sistemul de referință universal și, în continuare, vor fi supuse toate acelorași transformări.
- Transformarea de observare cu matricea  $M_v$ , corespunzătoare valorilor parametrilor  $x_v, y_v, z_v, \chi, \theta, \rho$  ai fiecărei imagini redată:

$$M_v = R_z(-\rho)R_x(-\theta)R_y(-\chi)T(-x_v, -y_v, -z_v),$$

- Transformarea de normalizare cu matricea:

$$M_N = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1.025 & -1.025 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Imaginile generate pe ecran mai necesită și alte transformări grafice, pe lângă cele prezentate până acum: transformarea în coordonate ecran 3D, prin care se efectuează o corespondență între fereastra de vizualizare și zona de afișare alocată pe ecran (numită poartă – *viewport*), precum și o transformare de rastru, prin care se generează mulțimea pixelilor afișați pe ecran. Aceste transformări vor fi descrise paragrafele următoare.



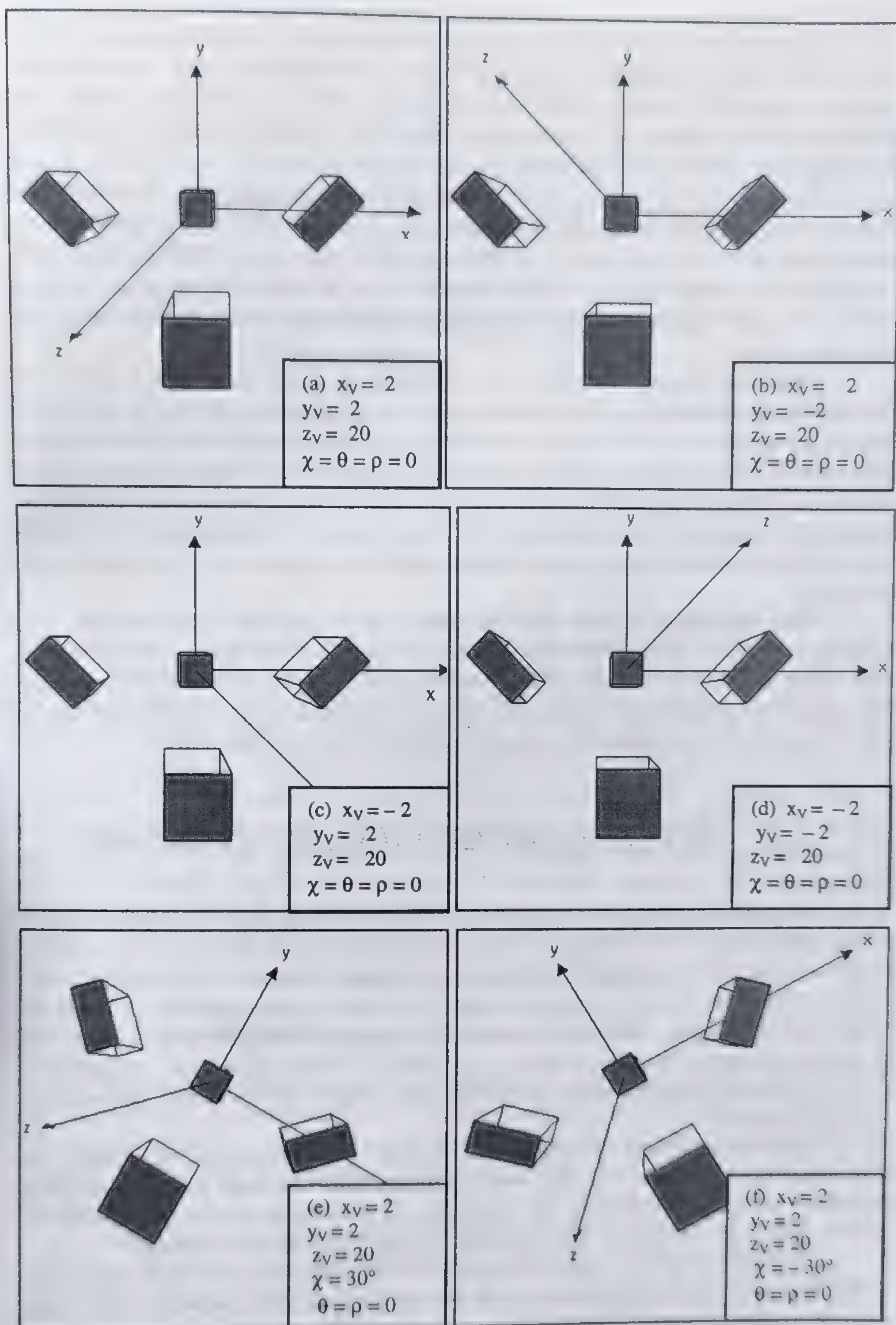


Fig. 4.7 Transformări de proiecție perspectivă.



Imaginile prezentate au fost captate din fereastra de afișare a programului care execută toate operațiile prezentate mai sus. Acest program a fost scris folosind biblioteca grafică OpenGL, care oferă funcții adecvate acestor operații de transformare și redare a imaginilor. Funcțiile OpenGL pentru realizarea transformărilor grafice vor fi descrise în capitolul 6.

În acest sistem de proiecție perspectivă s-a presupus că planul de vizualizare și planul de vizibilitate apropiată coincid. Deducerea matricei de normalizare pentru cazul general, în care cele două plane sunt diferite, este lăsată ca un exercițiu pentru cititori. De asemenea, tot ca un exercițiu, se poate deduce matricea de normalizare în situația în care centrul ferestrei de vizualizare nu se află pe axa  $z_v$ .

Prin specificarea sistemului de referință de observare, a volumului de vizualizare și a sistemului de proiecție (tipul de proiecție, centrul de proiecție, direcția de observare, fereastra de vizualizare) s-a definit un sistem de vizualizare care poate fi folosit pentru redarea obiectelor și a scenelor tridimensionale. Acest sistem este destul de flexibil și poate acoperi un număr mare de aplicații de grafică proiectivă (*Computer Aided Design* – CAD) și unele aplicații de realitate virtuală neimersivă de tip desktop, în care mediul grafic este generat pe monitorul unui calculator.

Dar restricțiile impuse acestui sistem de vizualizare (de exemplu, s-a considerat că centrul de proiecție coincide cu centrul sistemului de referință de observare) nu permit utilizarea lui în sisteme de realitate virtuală imersive sau semi-imersive în care este necesar un câmp de vizualizare de dimensiune mare. Astfel de situații se pot defini în sistemul de vizualizare standard PHIGS.

### 4.3 SISTEMUL DE VIZUALIZARE STANDARD

Imaginea generată pe un monitor, sau proiectată de un proiector pe un ecran, poate acoperi un câmp de vizualizare tipic de  $35-45^\circ$  vertical-orizantal. Pentru câmpuri de vizualizare mai mari se folosesc imagini multicanal, în care fiecare canal reprezintă un volum de vizualizare care se proiectează în fereastra de vizualizare a acestuia. Mai multe canale de imagine, care generează imagini pe monitoare juxtapuse sau proiectate pe un ecran de dimensiuni corespunzătoare, permit obținerea unor câmpuri de vizualizare extinse în sistemele de realitate virtuală imersive.

Pentru generarea imaginilor multicanal se utilizează un sistem de vizualizare care permite definirea tuturor parametrilor necesari pentru generarea simultană a mai multor ferestre de vizualizare. Un astfel de sistem de vizualizare este cel definit în standardele GKS și PHIGS, și va fi descris în continuare.

În sistemul de vizualizare standard PHIGS, sistemul de referință observator VRC (*view reference coordinate*) este definit prin punctul de observare VRP (*view reference point*), iar centrul de proiecție este diferit de acesta și este specificat prin punctul PRP (*projection reference point*). Planul de vizualizare (pe care se execută



proiecția imaginii) nu este în mod necesar perpendicular pe linia care unește centrul de proiecție cu centrul ferestrei de vizualizare, ceea ce permite realizarea proiecțiilor oblice. Fereastra de vizualizare este poziționată în orice loc în planul de vizualizare, ceea ce permite definirea și redarea simultană a mai multor imagini ale scenei. Sistemul de vizualizare PHIGS se specifică în trei etape. În prima etapă se definește sistemul de referință de observare; în cea de-a doua etapă se definește transformarea de normalizare, iar în ultima etapă se definește transformarea fereastră-poartă.

#### 4.3.1 DEFINIREA SISTEMULUI DE REFERINȚĂ DE OBSERVARE

Sistemul de referință de observare se definește prin următorii parametri specificați în sistemul de referință universal:

- Punctul de observare VRP (*view reference point*).
- Direcția normală la planul de vizualizare, VPN (*view plane normal*).
- Un vector a cărui proiecție determină sensul de prezentare a imaginii, VUP (*view up vector*), și care nu poate fi paralel cu vectorul VPN.

Punctul VRP, împreună cu vectorii VPN și VUP definesc sistemul de referință de observare ca un sistem orientat după regula mâinii drepte, cu axele notate U, V, N. N este vectorul VPN, normal la planul de vizualizare, iar UV este un plan paralel cu planul de vizualizare (fig. 4.8). Direcția de observare este orientată în sensul negativ al vectorului VPN.

În planul de vizualizare se definește fereastra de vizualizare, care este un dreptunghi orientat cu laturile paralele cu axele u și v ale unui sistem de referință bidimensional definit în acest plan. Acest sistem de referință are originea în punctul în care se proiectează (paralel cu direcția VPN) centrul sistemului de referință de observare (VRP) pe planul de vizualizare. Direcția axei v este paralelă cu proiecția (după direcția VPN) a vectorului VUP pe planul de vizualizare, astfel încât vectorul VUP specifică dacă scena este redată în direcție verticală (în sus sau în jos) sau pe direcție orizontală (spre dreapta sau spre stânga).

#### 4.3.2 DEFINIREA TRANSFORMĂRII DE NORMALIZARE

Parametrii transformării de normalizare definesc corespondența între volumul de vizualizare specificat în sistemul de referință observator și volumul canonic în sistemul de referință normalizat. Se specifică următorii parametri de definire a sistemului de referință normalizat relativ la sistemul de referință de observare:

- Tipul proiecției (paralelă sau perspectivă).
- Centrul de proiecție PRP (*projection reference point*).
- Distanța VPD (*view plane distance*) a planului de vizualizare față de centrul sistemului de referință de observare VRP.



- Distanța planului apropiat (*near*) și al planului depărtat de vizibilitate (*far*) față de centrul sistemului de referință de observare VRP.
- Limitele ferestrei de vizualizare,  $u_{min}$ ,  $u_{max}$ ,  $v_{min}$ ,  $v_{max}$ .

Planul de vizualizare este un plan paralel cu planul UV, aflat la distanța dată de valoarea VPD față de centrul sistemului de observare VRP. În planul de vizualizare se definește sistemul de coordonate plan uv, în care fereastra de vizualizare este un dreptunghi între limitele  $u_{min}$ ,  $u_{max}$ ,  $v_{min}$ ,  $v_{max}$ . Volumul de vizualizare se definește prin două plane paralele cu planul de vizualizare aflate la distanțele *near* și *far* de centrul sistemului de referință de observare VRP și planele care trec prin centrul de proiecție PRP și laturile ferestrei de vizualizare.

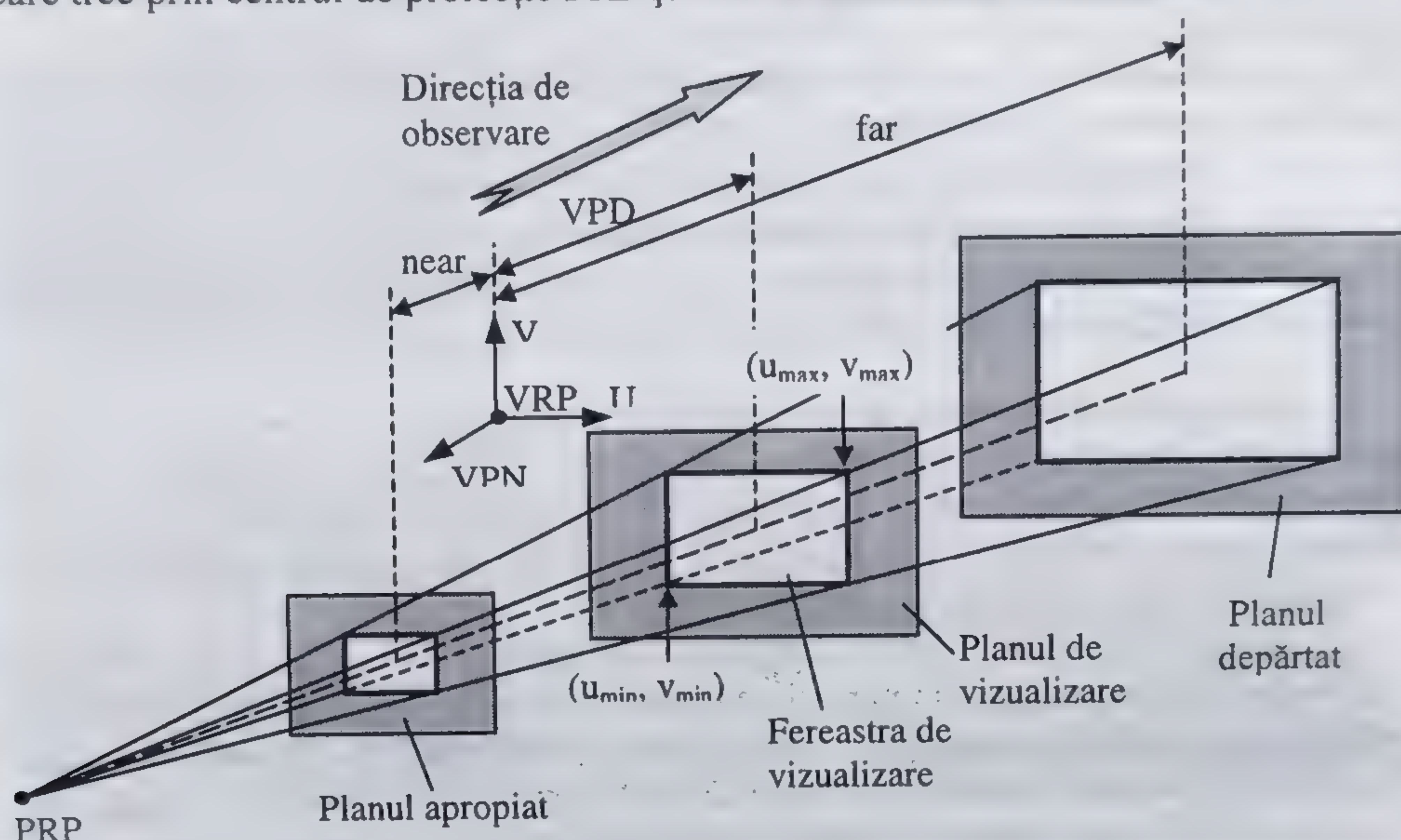


Fig. 4.8 Specificarea sistemului de vizualizare PHIGS în proiecția perspectivă.

Există două posibilități de poziționare a centrului de proiecție PRP față de fereastra de vizualizare:

- Dacă linia care unește centrul de proiecție cu centrul ferestrei de vizualizare este paralelă cu vectorul VPN, normal la planul de vizualizare, atunci se obține o proiecție perspectivă normală, echivalentă celei descrise anterior (fig. 4.9(a)).
- Dacă punctul PRP nu se află pe linia paralelă cu vectorul VPN care trece prin centrul ferestrei de vizualizare, atunci proiecția este o proiecție oblică (fig. 4.9(b)).

Implementarea sistemului de vizualizare PHIGS se face prin transformări geometrice succesive, la fel ca în exemplele prezentate până acum. Trebuie precizat că în sistemul PHIGS standard sistemul de referință de observare UVN este sistem drept, iar volumul de vizualizare și direcția de observare sunt orientate către axa - N.



O primă transformare geometrică este o schimbare de sistem de referință din sistemul de referință de observare UVN într-un sistem de referință notat Oxyz, cu centrul în centrul de proiecție PRP și cu axe paralele cu axele sistemului UVN (s-a ales această denumire pentru simplificarea notațiilor, dar acest sistem nu este sistemul de referință universal pentru care, în mod obișnuit, s-a folosit notația Oxyz). Această transformare se efectuează printr-o translație inversă celei definite de vectorul de poziție al punctului PRP.

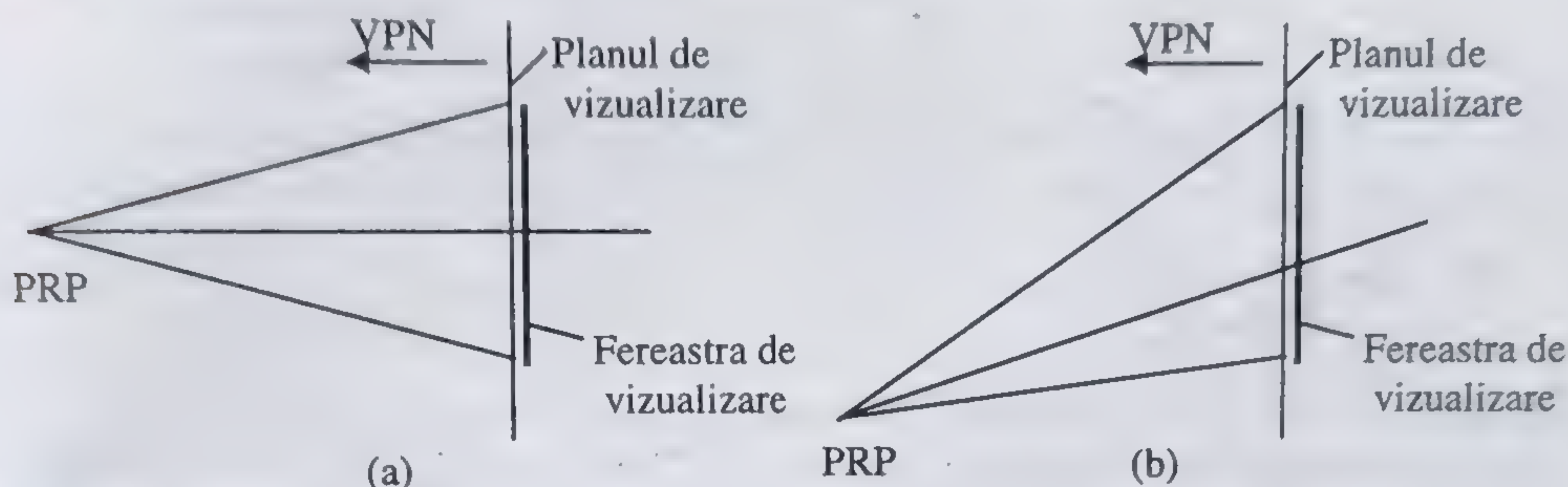


Fig. 4.9 (a) Proiecția normală; (b) Proiecție oblică.

Ca urmare a acestei transformări se modifică corespunzător valorile VPD, near, far,  $u_{\min}$ ,  $u_{\max}$ ,  $v_{\min}$ ,  $v_{\max}$ . Fie  $d$ ,  $n$ ,  $f$ ,  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$ , respectiv  $y_{\max}$ , valorile rezultate după translație. Aceste valori sunt numere reale și pot fi pozitive sau negative. Matricea de transformare de normalizare pornind din acest sistem Oxyz se poate descompune în produs de trei matrice:

$$M_N = M_{N2} M_{N1} M_{N0}$$

Matricele  $M_{N1}$  și  $M_{N2}$  au semnificații asemănătoare cu matricele dezvoltate pentru sistemul de vizualizare precedent. Cealaltă transformare componentă,  $M_{N0}$ , este o transformare de forfecare, prin care linia centrală a volumului de vizualizare (cea care unește centrul de proiecție PRP cu centrul ferestrei de vizualizare, fig. 4.10) se suprapune cu axa  $z$  a sistemului.

Transformarea de forfecare modifică valorile coordonatelor  $x$  și  $y$  cu valori proporționale cu  $z$ . Este simplu de observat că matricea:

$$M_{N0} = \begin{bmatrix} 1 & 0 & -(x_{\max} + x_{\min})/2d & 0 \\ 0 & 1 & -(y_{\max} + y_{\min})/2d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.14)$$

efectuează o simetrizare a volumului de vizualizare. De exemplu, intersecțiile muchiilor orizontale ale ferestrei de vizualizare cu planul  $x = 0$  se transformă astfel:

$$M_{N0} \begin{bmatrix} 0 & y_{\max} & d & 1 \end{bmatrix}^T = \begin{bmatrix} -(x_{\max} + x_{\min})/2 & (y_{\max} - y_{\min})/2 & d & 1 \end{bmatrix}^T$$

$$M_{N0} \begin{bmatrix} 0 & y_{\min} & d & 1 \end{bmatrix}^T = \begin{bmatrix} -(x_{\max} + x_{\min})/2 & -(y_{\max} - y_{\min})/2 & d & 1 \end{bmatrix}^T$$







A doua transformare componentă este transformarea de scalare  $M_{N1}$ , prin care volumul de vizualizare este transformat într-un volum simetric, cu unghiul de la vârful piramidei de  $90^\circ$ :

$$M_{N1} = \begin{bmatrix} 2d/(x_{\max} - x_{\min}) & 0 & 0 & 0 \\ 0 & 2d/(y_{\max} - y_{\min}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.15)$$

De exemplu, transformarea compusă  $M_{N1}M_{N0}$ , aplicată punctului din centrul ferestrei de vizualizare, îl transformă într-un punct aflat pe axa  $z$  negativă ( $d$  este negativ):

$$M_{N1}M_{N0}[(x_{\max} + x_{\min})/2 \quad (y_{\max} + y_{\min})/2 \quad d \quad 1]^T = [0 \quad 0 \quad d \quad 1]^T$$

Ultima componentă a transformării de normalizare transformă volumul de vizualizare în volum canonic (paralelipiped dreptunghic). Matricea acestei transformări,  $M_{N2}$ , este asemănătoare matricei descrise în paragraful precedent, dar se ține seama de faptul că planul de vizibilitate apropiat nu mai coincide cu planul de vizualizare:

$$M_{N2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-n) & -nf/(f-n) \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (4.16)$$

Sistemul normalizat rezultat este un sistem de referință stâng. În acest sistem se efectuează operațiile de decupare a obiectelor, după care secvența transformărilor necesare pentru generarea imaginii continuă cu transformarea în sistemul de referință ecran 3D.

## 4.4 SISTEMUL DE REFERINȚĂ ECRAN 3D

Transformarea în sistemul de referință ecran 3D efectuează o corespondență între fereastra din planul de vizualizare și zona de afișare alocată pe display numită poartă de afișare (*viewport*). Fereastra de vizualizare este definită într-un sistem de referință bidimensional în planul  $z_N = 0$  al sistemului de referință normalizat. Poarta de afișare este definită într-un sistem de referință bidimensional în planul  $z_S = 0$  al sistemului de referință ecran 3D și este, în general, o zonă rectangulară de dimensiuni egale cu numărul de pixeli corespunzători pe orizontală și pe verticală a zonei afișate (fig. 4.11).

Transformarea din sistemul de referință normalizat în sistemul ecran 3D lasă coordonata  $z$  nemodificată. Sistemul de referință ecran 3D astfel definit este un sistem de referință stâng, la fel ca și sistemul de referință normalizat.



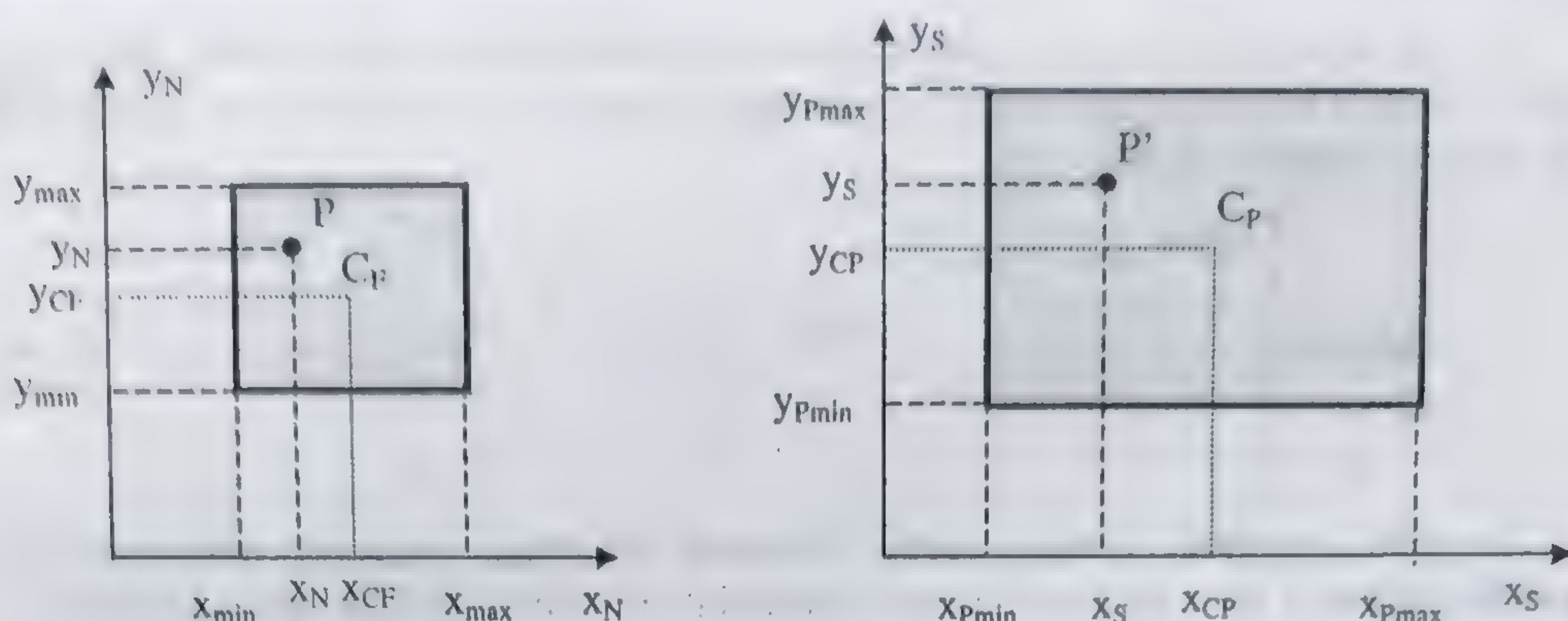


Fig. 4.11 Transformarea de la fereastra de vizualizare la poarta de afișare.

Pentru definirea transformării în sistemul de referință ecran 3D, se calculează mai întâi coordonatele centrului  $C_F$  al ferestrei și ale centrului  $C_P$  al porții:

$$\begin{aligned} x_{CF} &= (x_{\max} + x_{\min})/2; y_{CF} = (y_{\max} + y_{\min})/2 \\ x_{CP} &= (x_{P\max} + x_{P\min})/2; y_{CP} = (y_{P\max} + y_{P\min})/2 \end{aligned} \quad (4.17)$$

Corespondența dintre coordonatele unui punct  $P(x_N, y_N)$  din fereastră cu ale punctului  $P'(x_S, y_S)$  în poarta de afișare se formulează matematic astfel:

$$\frac{x_S - x_{CP}}{x_N - x_{CF}} = \frac{x_{P\max} - x_{P\min}}{x_{\max} - x_{\min}}, \quad \frac{y_S - y_{CP}}{y_N - y_{CF}} = \frac{y_{P\max} - y_{P\min}}{y_{\max} - y_{\min}}$$

Se definesc factorii de scalare ai transformării:

$$\begin{aligned} s_x &= \frac{x_{P\max} - x_{P\min}}{x_{\max} - x_{\min}} \\ s_y &= \frac{y_{P\max} - y_{P\min}}{y_{\max} - y_{\min}} \\ s_z &= 1 \end{aligned} \quad (4.18)$$

Rezultă:

$$\begin{aligned} x_S &= x_{CP} + s_x(x_N - x_{CF}) = s_x x_N + x_{CP} - s_x x_{CF} \\ y_S &= y_{CP} + s_y(y_N - y_{CF}) = s_y y_N + y_{CP} - s_y y_{CF} \\ z_S &= z_N \end{aligned} \quad (4.19)$$

Se poate deduce matricea de transformare ecran 3D prin compunerea a trei transformări succesive. Mai întâi se execută o translație, astfel încât centrul  $C_F$  al ferestrei să ajungă în origine, deci cu matricea de translație  $T(-x_{CF}, -y_{CF}, 0)$ .

Fereastra centrată în origine se scalează cu factorii de scară  $s_x, s_y$ , definiți de relația (4.18), astfel încât fereastra să ajungă la dimensiunea porții, iar



coordonata  $z$  se lasă nemodificată. Matricea de scalare este deci  $S(s_x, s_y, s_z)$ . Ultima transformare necesară este o translație, prin care centrul porții este adus în punctul  $C_P$ , deci cu matricea de translație  $T(x_{CP}, y_{CP}, 0)$ . Rezultă matricea de transformare din sistemul de referință normalizat în sistemul de referință ecran 3D:

$$M_{NS} = \begin{bmatrix} 1 & 0 & 0 & x_{CP} \\ 0 & 1 & 0 & y_{CP} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_{CP} \\ 0 & 1 & 0 & -y_{CP} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.20)$$

$$M_{NS} = \begin{bmatrix} s_x & 0 & 0 & x_{CP} - s_x x_{CP} \\ 0 & s_y & 0 & y_{CP} - s_y y_{CP} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ w_S \end{bmatrix} = M_{NS} \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix}$$

Din aceste relații rezultă  $Z_S = Z_N$  și  $w_S = w_N$ . În sistemul de referință ecran 3D se trece de la coordonatele omogene  $X_S, Y_S, Z_S, w_S$ , la coordonatele tridimensionale  $x_S, y_S, z_S$  prin împărțirea cu  $w_S$ .

Planul  $z_S = 0$  din sistemul de referință ecran 3D este planul de proiecție. Coordonatele  $x_S$  și  $y_S$  reprezintă coordonatele proiecției ortografice în planul  $z_S = 0$  (care este planul de porții de afișare). Transformarea de proiecție perspectivă din sistemul de referință de observare a devenit o transformare de proiecție ortografică în sistemul de referință ecran 3D. Coordonata  $z_S$  în sistemul de referință ecran 3D este utilizată în algoritmi de eliminare a suprafețelor ascunse. Volumul de vizualizare în sistemul ecran 3D este un paralelipiped dreptunghic cu baza un dreptunghi de dimensiuni egale cu dimensiunile porții afișate și înălțimea egală cu 1 ( $z_S \in [0,1]$ ) (fig. 4.12).

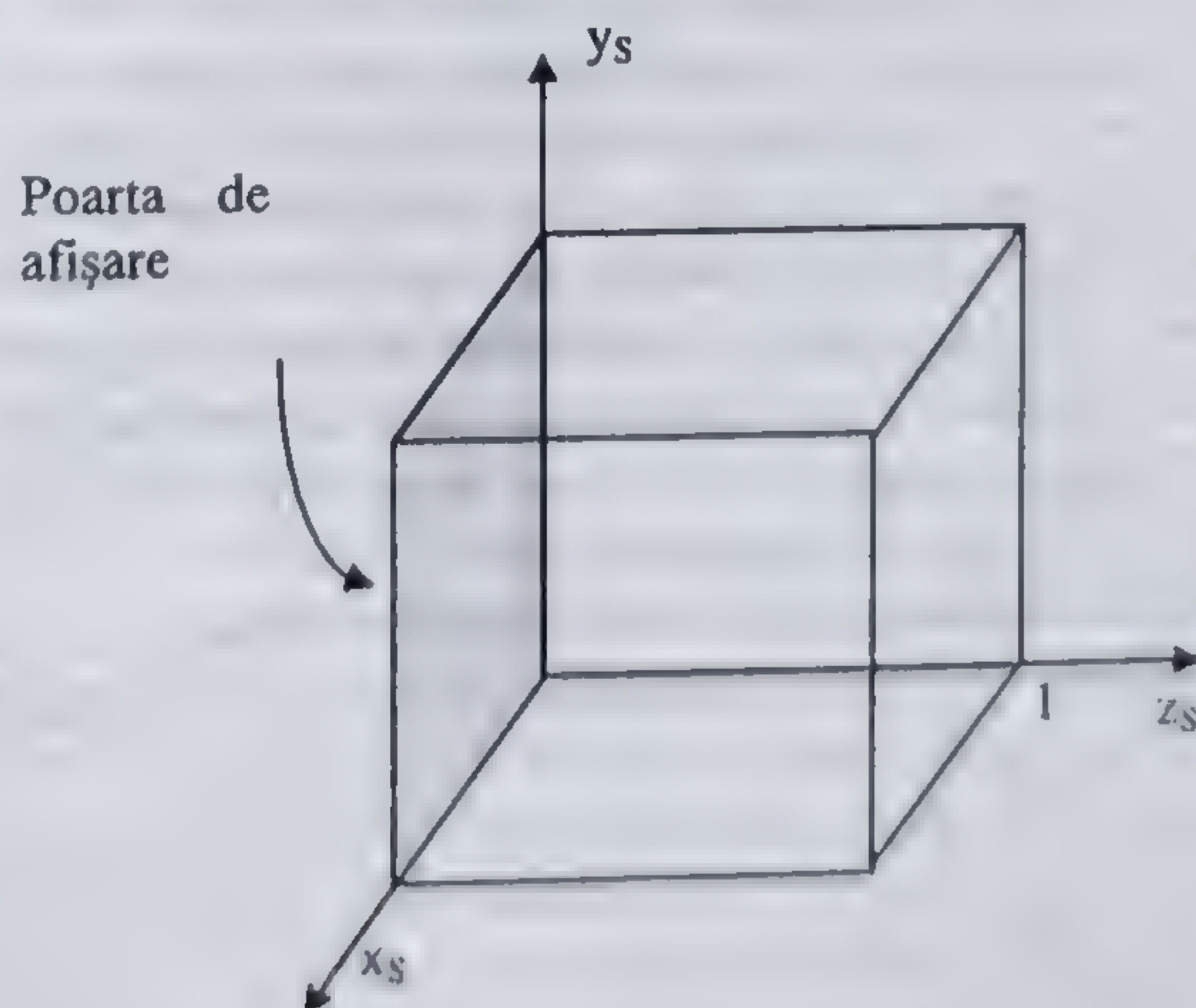


Fig. 4.12 Volumul de vizualizare în sistemul de referință ecran 3D.



## 4.5 DECUPAREA OBIECTELOR

Într-un sistem în care este definit punctul de observare și un volum de vizualizare, nu interesează decât obiectele din interiorul acestui volum, a căror proiecție se generează în fereastra de vizualizare. De aceea, în vizualizarea scenelor tridimensionale se execută operația de decupare la volumul de vizualizare (*clipping*), prin care se rețin numai obiectele sau părțile din obiecte cuprinse în acest volum.

Operația de decupare la volumul de vizualizare poate fi definită astfel: fiind dat un obiect tridimensional și un volum de vizualizare, pot exista trei posibilități:

- Obiectul este complet în interiorul volumului și, în acest caz, el este proiectat în fereastră și redat pe display.
- Obiectul este complet în afara volumului de vizualizare și, în acest caz, el poate fi ignorat.
- Obiectul intersectează volumul de vizualizare; în acest caz se decupează obiectul și partea vizibilă (cuprinsă în interiorul volumului) este proiectată în fereastra de vizualizare și redată pe display.

Decuparea obiectelor tridimensionale la volumul de vizualizare se execută în sistemul de referință normalizat în coordonate omogene, înainte de împărțirea cu  $w$ . Operația de decupare în spațiul tridimensional reprezintă o extindere a decupării în planul bidimensional, de aceea se va prezenta mai întâi decuparea în plan.

### 4.5.1 DECUPAREA ÎN PLAN

Fiind dat un poligon oarecare în plan și o regiune de decupare, problema decupării poligonului este de a reține numai acea parte din poligon care se află în interiorul regiunii de decupare. Se consideră cazul cel mai frecvent întâlnit în care regiunea de decupare este un dreptunghi cu laturile paralele cu axele de coordonate. Această regiune de decupare corespunde ferestrei de vizualizare.

Majoritatea algoritmilor de decupare, atât în plan cât și în spațiu, se bazează pe algoritmul de decupare Sutherland-Hodgman [Suth74]. Algoritmul Sutherland-Hodgman împarte planul bidimensional în nouă regiuni, după poziția față de regiunea dreptunghiulară de decupare dată prin ecuațiile a patru drepte:  $x = x_{min}$ ;  $x = x_{max}$ ;  $y = y_{min}$ ;  $y = y_{max}$  (fig. 4.13).

Fiecărei regiuni îi corespunde un cod de vizibilitate (cunoscut sub numele de cod Sutherland-Cohen) reprezentat pe 4 biți,  $b_0, b_1, b_2, b_3$ , definiți astfel:

$$b_0 = \text{semn}(x - x_{min})$$

$$b_1 = \text{semn}(x_{max} - x)$$

$$b_2 = \text{semn}(y - y_{min})$$

$$b_3 = \text{semn}(y_{max} - y)$$

În continuare, codurile de vizibilitate se reprezintă printr-o secvență de cifre binare.



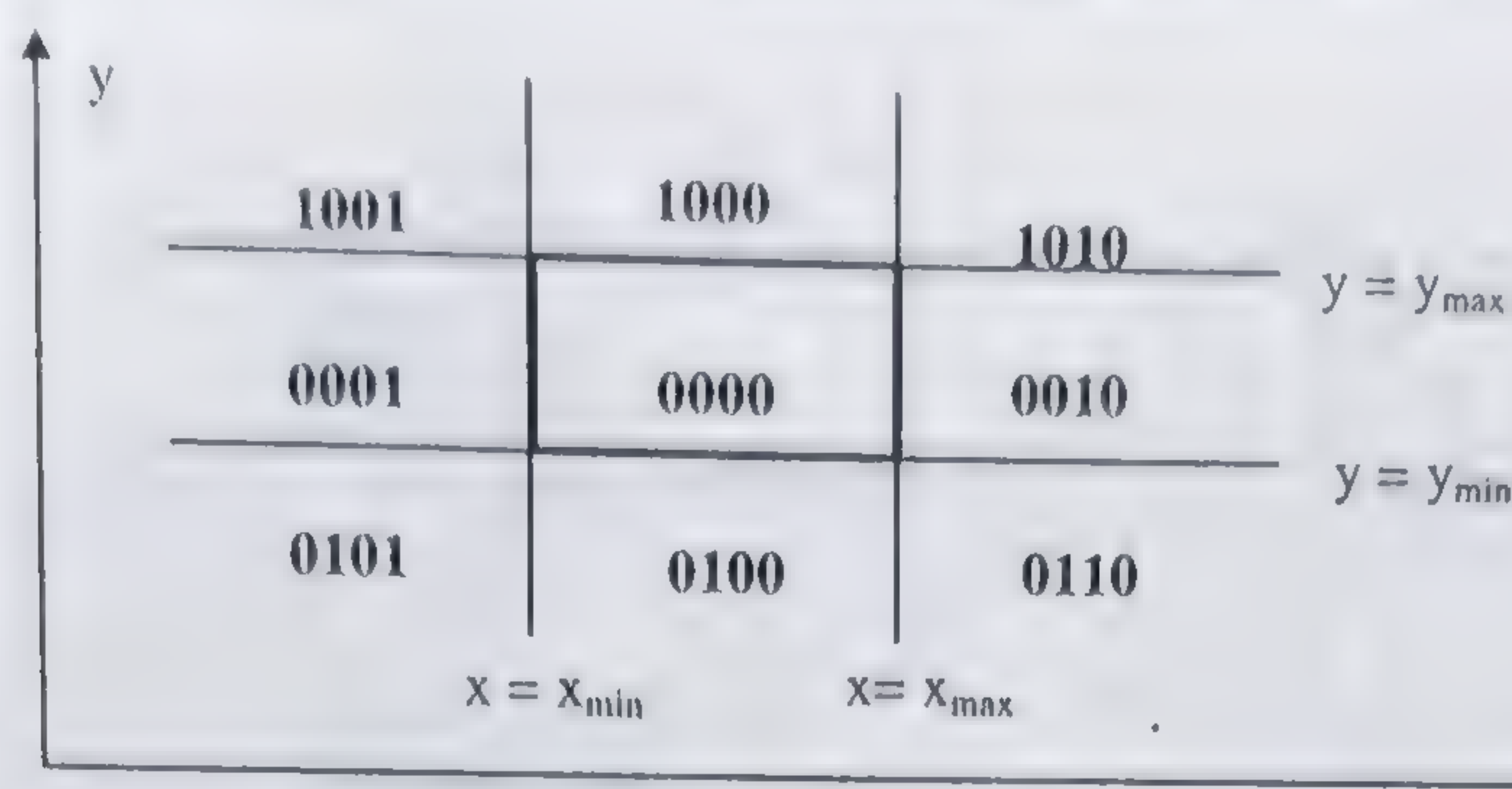


Fig. 4.13 Codurile de vizibilitate Sutherland-Cohen.

Un poligon oarecare din plan poate fi complet inclus în regiunea de decupare, poate fi complet în afara regiunii de decupare sau o poate intersecta. Folosind codurile de vizibilitate ale vârfurilor poligonului se pot identifica direct (fără să se execute calcule de intersecții) două situații ale poligonului față de regiunea de decupare:

- *Acceptare sigură* (fig. 4.14 (a)). Dacă toate vârfurile poligonului se află în regiunea de decupare, atunci poligonul este vizibil în întregime. Fie  $C_1, C_2, \dots, C_n$ , codurile de vizibilitate ale vârfurilor poligonului. Pentru testul de acceptare sigură se calculează reuniunea acestor coduri (operația OR):  $SC = C_1 | C_2 | \dots | C_n$ . Dacă  $SC = 0$ , atunci poligonul este sigur vizibil.
- *Rejecție sigură* (fig. 4.14 (b)). Dacă toate vârfurile unui poligon se află într-unul din semiplanele invizibile ( $x < x_{\min}$ ;  $x > x_{\max}$ ;  $y < y_{\min}$ ;  $y > y_{\max}$ ), atunci tot poligonul este sigur invizibil. Pentru testul de rejecție sigură se calculează intersecția codurilor de vizibilitate (operația AND):  $PC = C_1 \& C_2 \& \dots \& C_n$ . Dacă  $PC \neq 0$ , atunci poligonul este sigur invizibil deoarece toate punctele se află în semiplanul invizibil corespunzător bitului diferit de 0 din PC.

Dacă nu este îndeplinită nici condiția de acceptare sigură, nici condiția de rejecție sigură, atunci decuparea se calculează prin intersecția laturilor poligonului cu dreptele care mărginesc regiunea de decupare (fig. 4.14 (c)).

Dreapta (infinită) corespunzătoare unei laturi a regiunii de decupare împarte planul în două semiplane: semiplanul (regiunea) vizibil, care se află de aceeași parte a dreptei ca și regiunea de decupare, și semiplanul invizibil, care se află de partea opusă față de regiunea de decupare. De exemplu, pentru latura  $x = x_{\min}$ , semiplanul  $x < x_{\min}$  este invizibil, iar semiplanul  $x \geq x_{\min}$  este vizibil.

În algoritmul de decupare Sutherland-Hodgman, decuparea unui poligon față de o regiune dată se efectuează într-un număr de pași egal cu numărul de laturi ale regiunii de decupare. În fiecare pas se decupează poligonul (inițial sau provenit dintr-un pas anterior) cu o latură a regiunii de decupare: se elimină acea porțiune din poligon care se află în semiplanul invizibil corespunzător laturii și se reține numai porțiunea vizibilă.



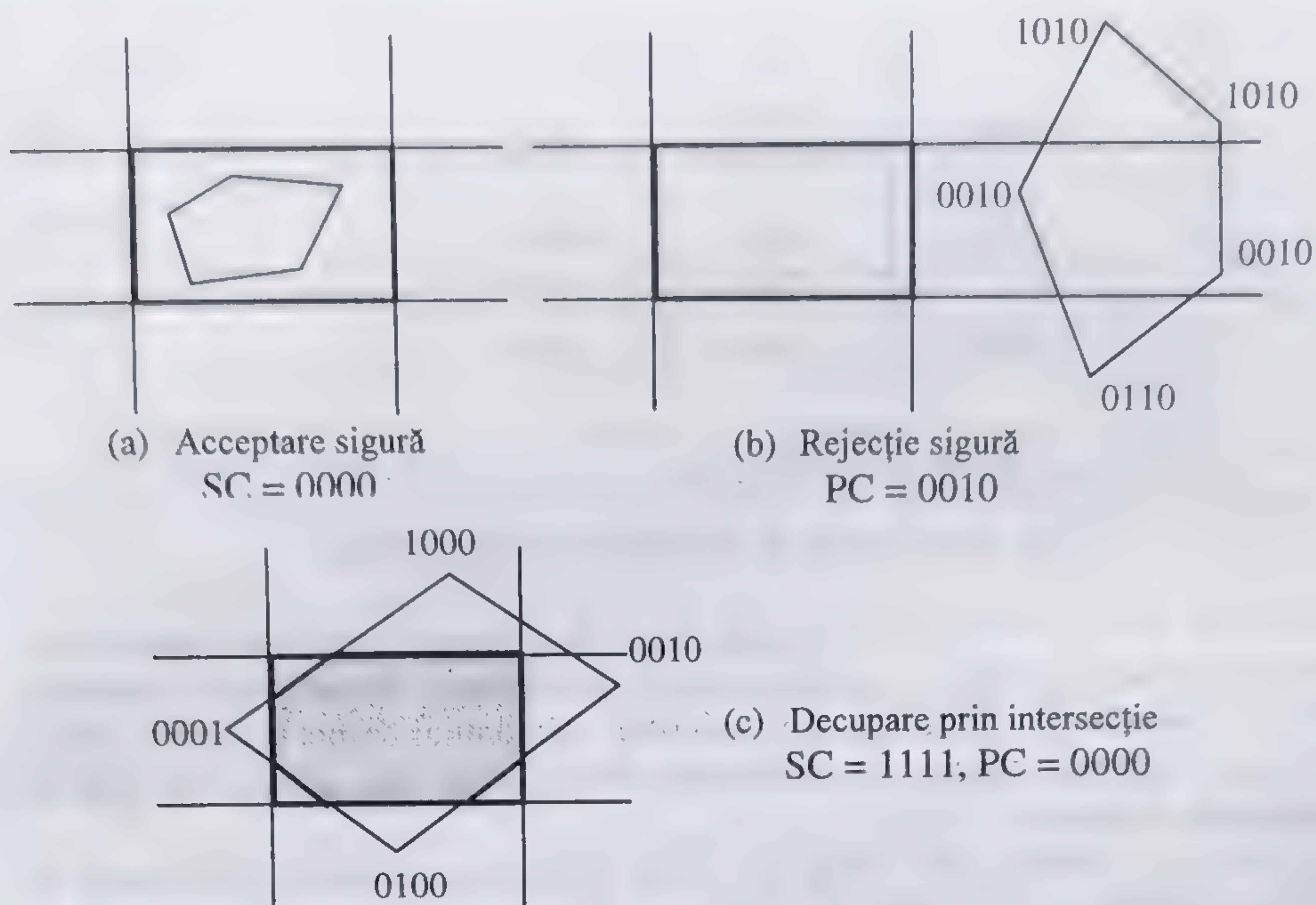


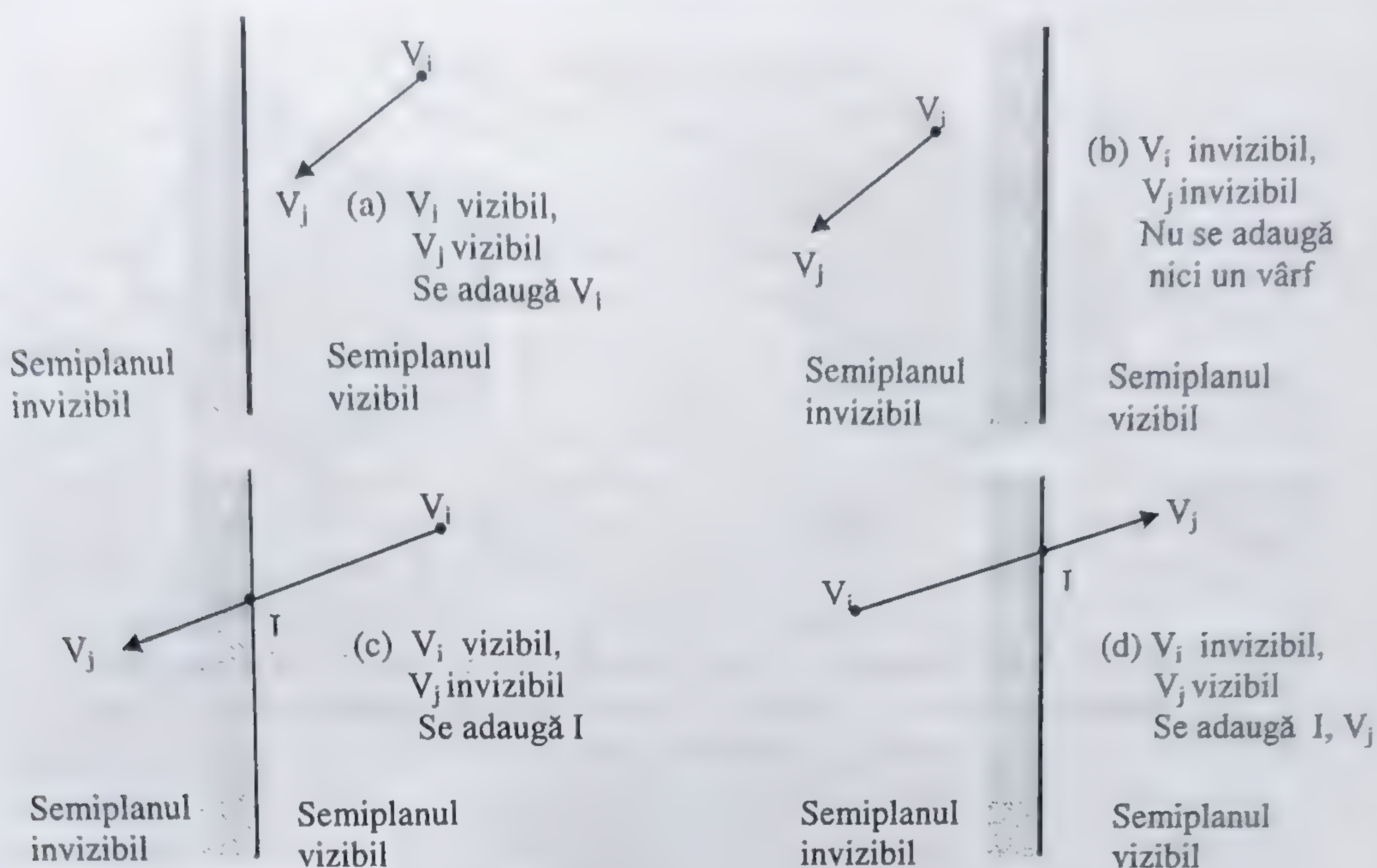
Fig. 4.14 Teste de vizibilitate pentru decuparea în plan.

Poligonul se reprezintă printr-o listă ordonată de vârfuri  $V_1V_2...V_n$ , și fiecare pereche de două vârfuri consecutive ( $V_i V_{i+1}$ ), pentru  $i < n$ , și ( $V_nV_1$ ) reprezintă o latură a poligonului ca un segment de dreaptă orientat (vector).

În fiecare pas de decupare se pornește de la un poligon dat printr-o listă ordonată de vârfuri și se creează lista vârfurilor poligonului rezultat prin decupare relativ la dreapta suport a unei laturi a regiunii de decupare. Inițial, lista vârfurilor poligonului rezultat este considerată vidă. Un segment orientat  $V_iV_j$  se poate afla într-una din patru situații posibile față de o latură de decupare (fig. 4.15).

- Dacă ambele vârfuri se află în semiplanul vizibil corespunzător laturii de decupare, atunci se adaugă în listă vârful  $V_j$  (fig. 4.15 (a)).
- Dacă ambele vârfuri se află în semiplanul invizibil, atunci nu se adaugă nici un vârf în listă (fig. 4.15 (b)).
- Dacă primul vârf al segmentului ( $V_i$ ) se află în semiplanul vizibil, iar al doilea ( $V_j$ ) se află în semiplanul invizibil (segmentul este orientat de la semiplanul vizibil către semiplanul invizibil), atunci se calculează intersecția  $I$  dintre segment și dreapta de decupare și se adaugă intersecția  $I$  în listă (fig. 4.15(c)).
- Dacă primul vârf al segmentului ( $V_i$ ) se află în semiplanul invizibil, iar al doilea ( $V_j$ ) se află în semiplanul vizibil (segmentul este orientat de la semiplanul invizibil către semiplanul vizibil), atunci se calculează intersecția  $I$  dintre segment și dreapta de decupare. În lista vârfurilor se adaugă intersecția  $I$  și apoi vârful  $V_j$  (fig. 4.15 (d)).





**Fig. 4.15** Intersecția dintre o latură a poligonului și o dreaptă de decupare.

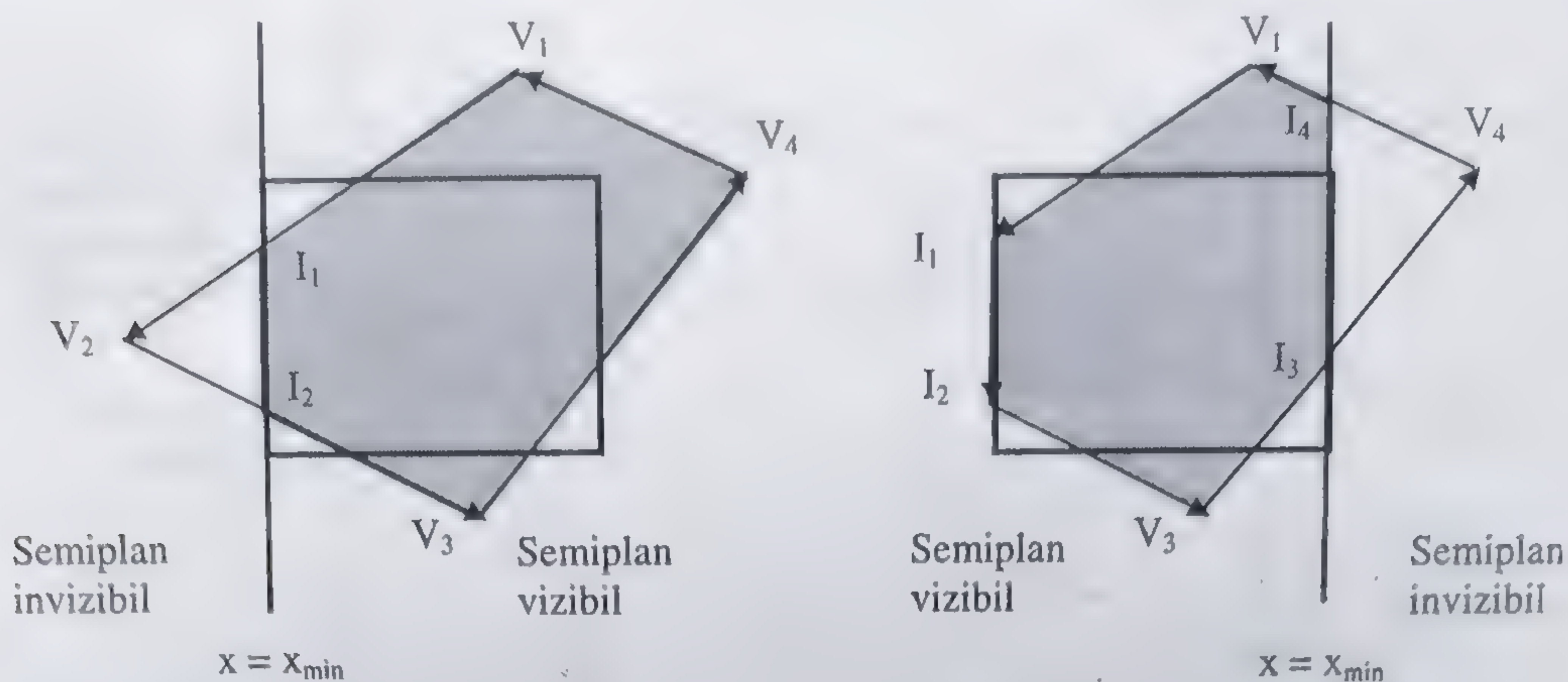
În primul pas, poligonul inițial  $V_1V_2...V_n$  se decupează relativ la latura  $x = x_{\min}$  prin parcurgerea în ordine a tuturor segmentelor orientate ale poligonului (fig. 4.16(a)). Poligonul rezultat se decupează relativ la următoarea latură a regiunii de decupare (fig. 4.16 (b)). După parcurgerea unui număr de pași egal cu numărul de laturi ale regiunii de decupare se obține poligonul complet decupat (fig. 4.16(d)).

În fig.4.16 este ilustrat algoritmul de decupare Sutherland-Hodgman al unui poligon convex relativ la o regiune de decupare rectangulară, dar acest algoritm poate fi folosit în cazul general, al decupării oricărui tip de poligon (convex, concav sau cu găuri) relativ la o regiune de decupare oarecare.

Decuparea (în plan sau în spațiu) poate fi executată în orice operație care necesită construirea de poligoare sau obiecte tridimensionale noi prin calcul, pornind de la obiecte de bază, care sunt intersectate cu alte obiecte sau suprafețe.

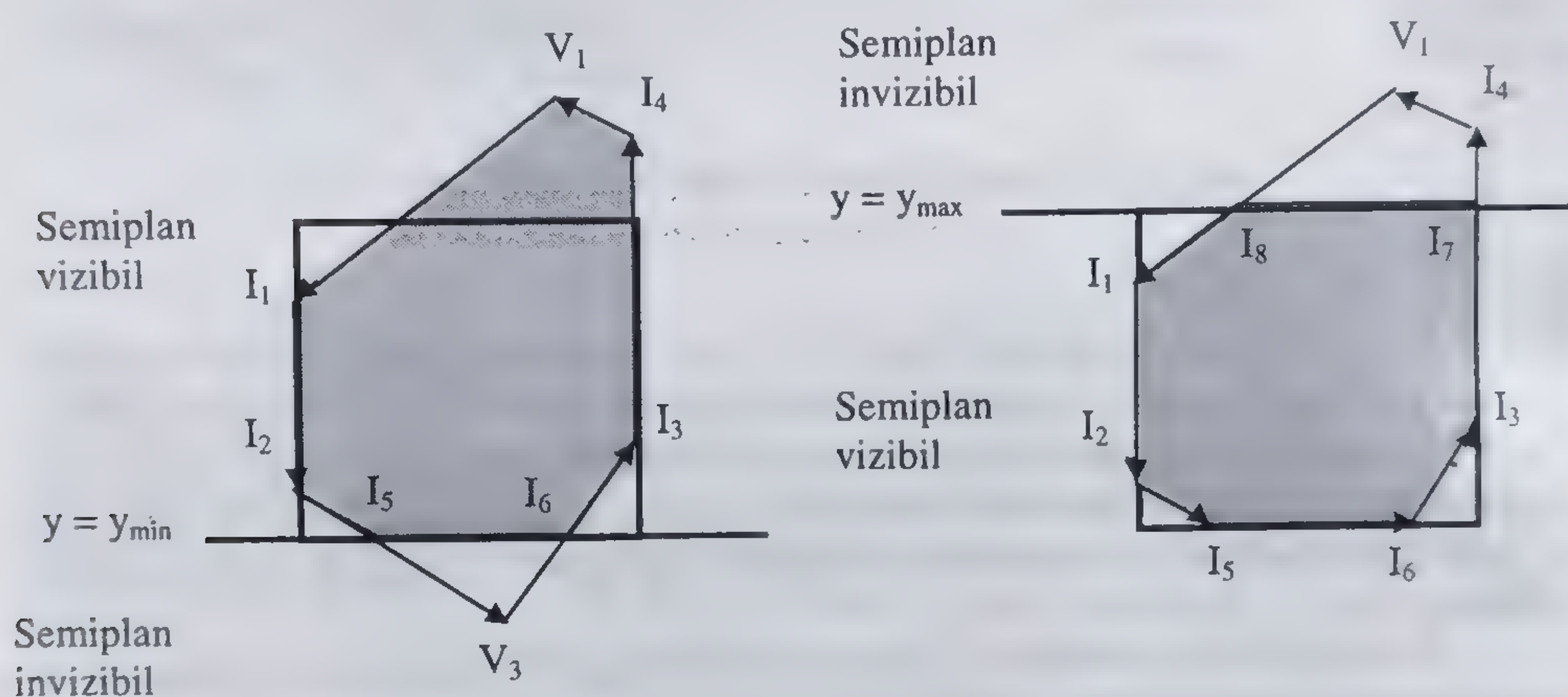
În generarea imaginilor bidimensionale, decuparea, ca operație componentă a secvenței de transformări de vizualizare, se efectuează relativ la fereastra de vizualizare, care este o regiune rectangulară. Poligoanele sau liniile rezultate, limitate la dimensiunea ferestrei, sunt transformate în sistemul de referință al porții de afișare și redată pe display. Dacă nu se efectuează decuparea poligoanelor la limitele ferestrei, coordonatele vârfurilor în sistemul de referință al porții de afișare depășesc limitele porții, iar rezultatul conversiei de rastru este incorect.





(a) Poligonul  $V_1V_2V_3V_4$  decupat în semiplanul  $x \geq x_{\min}$

(b) Poligonul  $I_1I_2V_3V_4V_1$  decupat în semiplanul  $x \leq x_{\max}$



(c) Poligonul  $I_2V_3I_3I_4V_1$  decupat în semiplanul  $y \geq y_{\min}$

(d) Poligonul  $I_2I_5I_6I_3I_4V_1$  decupat în semiplanul  $y \geq y_{\min}$

**Fig. 4.16** Decuparea unui poligon în algoritmul Sutherland-Hodgman.  
Poligonul rezultat este:  $I_5 I_6 I_3 I_7 I_8 I_1 I_2$

Se poate efectua o decupare a poligoanelor și liniilor și în poarta de afișare, prin testul individual al coordonatelor fiecărui pixel și rejectarea acelor care nu se încadrează în limitele porții de afișare. Acesta este însă un procedeu deosebit de costisitor ca timp de execuție și nu se pune problema utilizării lui în grafica interactivă și în realitatea virtuală.



### 4.5.2 DECUPAREA SUPRAFETELOR RELATIV LA VOLUMUL DE VIZUALIZARE

Algoritmii de decupare a obiectelor în spațiul tridimensional pot fi considerați ca o extensie a algoritmilor de decupare din plan, prin înlocuirea regiunii de decupare plane cu un volum de decupare și efectuarea calculului de intersecție a unei suprafețe în spațiu cu planele volumului de decupare.

Decuparea la volumul de vizualizare este un caz particular al decupării spațiale, în care volumul de decupare este volumul de vizualizare (trunchiul de piramidă de vizualizare) definit în sistemul de referință de observare. În continuare se va analiza modul de execuție a decupării pentru sistemul de vizualizare definit în fig. 4.5 și se va evidenția de ce decuparea se execută cel mai eficient în sistemul de referință normalizat, afirmație care nu a fost încă argumentată.

Volumul de decupare se definește prin șase plane în sistemul de referință de observare, ale căror ecuații sunt date de relațiile (4.9). Fiecare plan împarte spațiul tridimensional în două semispații, semispațiul vizibil, aflat de aceeași parte a planului ca și volumul de decupare, și semispațiul invizibil, în partea opusă a volumului. Decuparea unei suprafețe în raport cu acest plan se efectuează asemănător cu decuparea unui poligon față de o latură de decupare: fiecare latură orientată introduce zero, unul sau două vârfuri în lista de vârfuri a suprafeței rezultate, în funcție de orientare și de intersecția ei cu planul. Un vârf este vizibil în raport cu un plan dacă se află în semispațiul vizibil determinat de planul dat. De exemplu, un vârf este vizibil în raport cu planul  $y_v = h z_v / d$ , dacă  $y_v \leq h z_v / d$  (fig. 4.17 (a)). În mod asemănător, decuparea relativ la celelate plane ale volumului de decupare definit în sistemul de referință de observare stâng necesită testele:

$$y_v \geq -\frac{h}{d} z_v; x_v \leq \frac{g}{d} z_v; x_v \geq -\frac{g}{d} z_v; z_v \geq d; z_v \leq f \quad (4.21)$$

Aceste teste implică operații asupra coordonatelor vârfurilor, operații care în formă matriceală înseamnă înmulțirea cu matricea  $M_{N1}$ , componentă a matricei de normalizare. Așa cum se poate observa în fig. 4.17(b), matricea  $M_{N1}$  transformă sistemul de referință de observare într-un sistem de referință  $O'x'y'z'$ , în care volumul de vizualizare (deci volumul de decupare) este un trunchi de piramidă regulată cu unghiul de la vârful piramidei de  $90^\circ$ . Relațiile (4.21) de testare a vizibilității unui punct în sistemul de referință de observare sunt echivalente cu următoarele relații în sistemul  $O'x'y'z'$ :

$$x' \leq z'; x' \geq -z'; y' \leq z'; y' \geq -z'; z' \leq f; z' \geq d \quad (4.22)$$

Dacă s-ar executa decuparea în sistemul de referință intermediar  $O'x'y'z'$ , atunci toate punctele din sistemul de referință de observare se transformă mai întâi cu matricea  $M_{N1}$ , pentru trecerea în sistemul  $O'x'y'z'$ , apoi se execută decuparea, iar după decupare, vârfurile vizibile se transformă cu matricea  $M_{N2}$ , pentru trecere în sistemul de referință normalizat și proiecție în fereastra de vizualizare (fig. 4.17(c)). Aplicarea separată a două transformări, mai întâi prin matricea  $M_{N1}$  și apoi prin matricea  $M_{N2}$  este costisitoare și nenenecară, dat fiind că decuparea se poate face direct în sistemul normalizat (fig. 4.17 (c)).



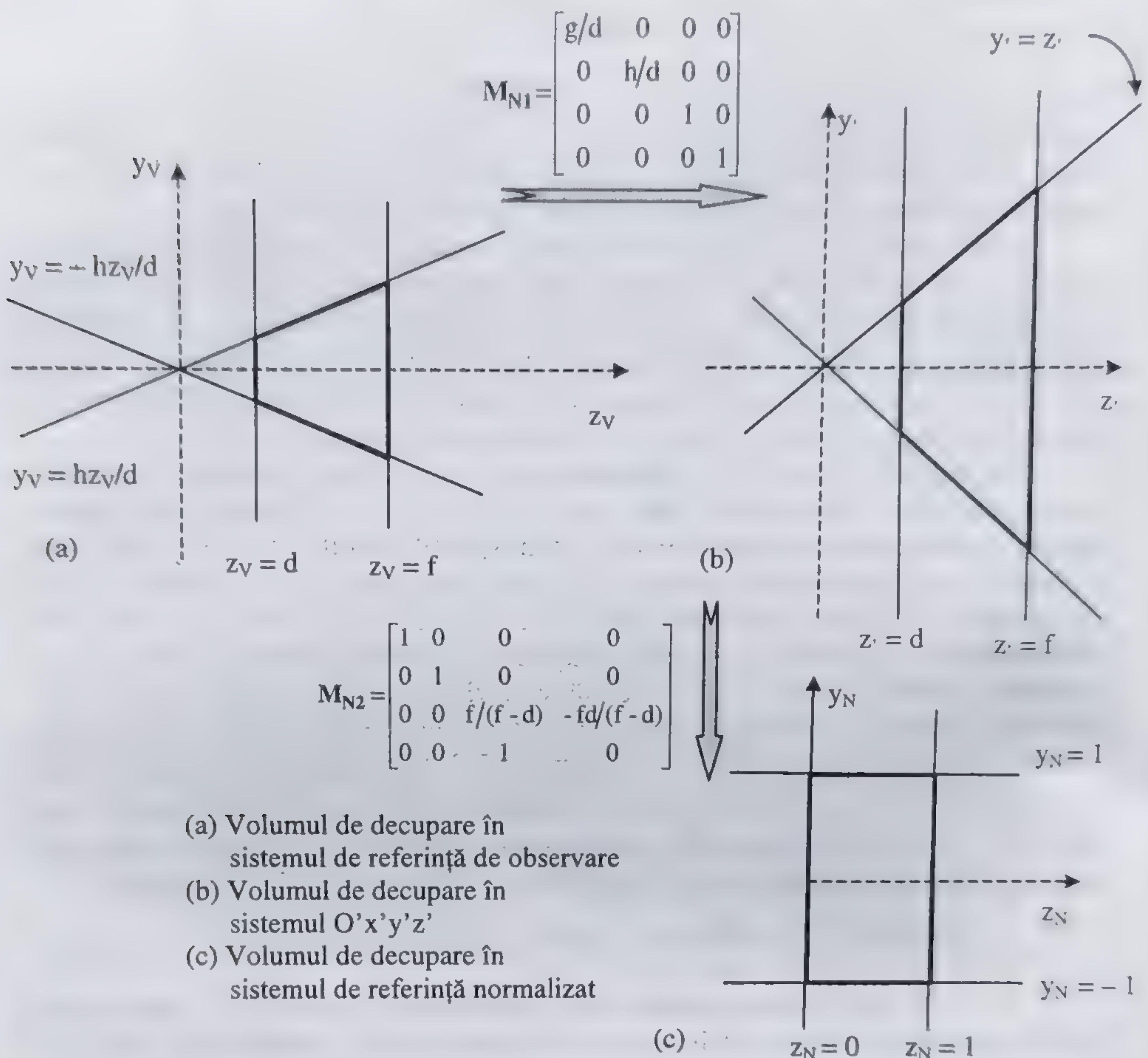


Fig. 4.17. Volumul de decupare în sistemul de referință normalizat.  
(Proiecție în planul  $x = 0$ ).

În sistemul de referință normalizat volumul de decupare este un paralelipiped dreptunghic cu suprafețele paralele cu planele sistemului de coordonate. Testele de vizibilitate ale unui punct în raport cu planele volumului de decupare sunt:

$$x_N \leq 1; \quad x_N \geq -1; \quad y_N \leq 1; \quad y_N \geq -1; \quad z_N \leq 1; \quad z_N \geq 0; \quad (4.23)$$

Aceste teste se pot efectua direct în coordonatele omogene în sistemul normalizat, fără să mai fie necesară împărțirea cu  $w_N$ :

$$\begin{aligned} X_N &\leq w_N; \quad X_N \geq -w_N; \\ Y_N &\leq w_N; \quad Y_N \geq -w_N; \\ Z_N &\leq w_N; \quad Z_N \geq 0; \end{aligned} \quad (4.24)$$



Codul de vizibilitate  $C$  al unui vârf în coordonate normalizate omogene se exprimă pe 6 biți și fiecare bit se poziționează în funcție de rezultatul unuia din testele din relația (4.24):

$$b_0 = \text{semn}(X_N - w_N)$$

$$b_1 = \text{semn}(w_N - X_N)$$

$$b_2 = \text{semn}(Y_N - w_N)$$

$$b_3 = \text{semn}(w_N - Y_N)$$

$$b_4 = \text{semn}(Z_N - w_N)$$

$$b_5 = \text{semn}(Z_N)$$

Spațiul omogen normalizat se împarte în 27 de regiuni după poziția lor față de planele care definesc volumul canonic de vizualizare. În interiorul volumului canonic, toate punctele au codul de vizibilitate cu toți biții egali cu 0 ( $C = 000000$ ). În toate celelalte regiuni, codul de vizibilitate este diferit de 0, având unul sau mai mulți biți egali cu 1.

Algoritmul Sutherland-Hodgman de decupare în spațiu se desfășoară în mod asemănător cu algoritmul de decupare în plan. Pentru o suprafață dată prin vârfurile ei în spațiu, decuparea relativ la un volum dat prin ecuațiile planelor sale se efectuează în modul următor:

- (1) Se calculează codurile de vizibilitate ale vârfurilor suprafeței.
- (2) Se efectuează testele de acceptare și rejecție sigură. Pentru suprafețele care nu îndeplinesc nici unul din aceste teste se continuă cu decuparea prin intersecția cu volumul de decupare.
- (3) Decuparea prin intersecție a unei suprafețe se efectuează într-un număr de pași egal cu numărul de plane ale volumului de decupare. În fiecare pas se obține o nouă suprafață, din care a fost eliminată porțiunea invizibilă în raport cu planul de decupare respectiv.

După efectuarea decupării, se calculează coordonatele  $x_N$ ,  $y_N$ ,  $z_N$  ale vârfurilor suprafețelor vizibile rezultate (prin împărțirea cu  $w_N$ ), iar toate celelalte vârfuri (ale suprafețelor invizibile) sunt ignorate. Acest lucru înseamnă că operația de împărțire omogenă se execută pentru un număr strict necesar de vârfuri, ceea ce reprezintă un câștig în viteza de execuție.

Se pune problema dacă nu ar fi mai simplu să fie efectuată decuparea în plan, deci după calculul proiecțiilor suprafețelor, dat fiind că algoritmul de decupare în plan este mai simplu. Acest lucru ar însemna ca toate suprafețele să fie proiectate pe planul de vizualizare, după care să se efectueze decuparea relativ la fereastra de vizualizare. Această soluție este însă inacceptabilă, datorită faptului că în fereastra de proiecție se proiectează atât punctele (vârfuri ale suprafețelor) aflate în volumul de vizualizare, cât și cele aflate într-un volum simetric cu acesta față de planul  $z_v = 0$ , care se află în spatele observatorului.

Tratarea tuturor acestor suprafețe proiectate în fereastra de vizualizare dar invizibile consumă un timp de execuție în mod inutil. De aceea, decuparea obiectelor la volumul de vizualizare nu se poate executa corect și eficient decât în spațiul tridimensional, și anume în coordonate normalizate omogene.



### 4.5.3 ELIMINAREA OBIECTELOR AFLATE ÎN EXTERIORUL VOLUMULUI DE VIZUALIZARE

Se poate detalia modul de execuție a decupării relativ la volumul de vizualizare. Algoritmul de bază se referă la decuparea unei suprafețe în spațiu, dată ca o succesiune de vârfuri și, implicit, o succesiune de segmente de dreaptă orientate (laturile suprafeței), relativ la un volum de decupare, care este volumul canonic în coordonate omogene normalizate.

Decuparea punctelor în spațiu este un simplu test de vizibilitate (ecuațiile 4.24), iar decuparea unei linii este un caz particular (cu  $n = 2$ ) al algoritmului de decupare a unei suprafețe.

Decuparea unui obiect tridimensional, modelat ca o colecție de fețe (suprafețe plane), se reduce la decuparea fiecărei fețe în parte, iar obiectul rezultat este compus din toate fețele sale vizibile și părțile rezultate prin decupare.

O creștere a eficienței în operația de decupare se poate obține folosind teste de rejecție și de acceptare sigură nu numai pentru fiecare suprafață în parte, dar și pentru un obiect tridimensional în întregime.

Procedeul de calcul al reuniunii (SC) și al intersecției (PC) codurilor de vizibilitate ale unui obiect tridimensional depinde de modul de reprezentare a acestuia. Pentru obiectele tridimensionale modelate prin fețe date ca listă de indecși la vârfuri se calculează direct valorile SC și PC ale celor  $N$  vârfuri ale obiectului. Pentru obiecte reprezentate prin fețe separate, se pot colecta codurile de vizibilitate ale fețelor componente și calcula reuniunea SC și intersecția PC. Funcția de acceptare sau rejecție sigură (Culling) a unui obiect tridimensional este prezentată în pseudocod asemănător limbajului C astfel:

```
int Culling() {
    PC = 0x3F;
    SC = 0;
    for (i=0; i<N; i++){
        SC = SC | Ci;
        PC = PC & Ci;
    }
    if (PC) {
        Obiect sigur invizibil, va fi rejectat;
        return 0;
    }
    else if (SC){
        Obiect sigur vizibil în întregime;
        Nu se va mai testa fiecare suprafață;
        return 1;
    }
    else {
        Se va decupa fiecare suprafață;
        return -1;
    }
}
```



Ce anume se execută în fiecare din situații depinde, evident, de modul de organizare al programului și de biblioteca grafică folosită. În orice caz, dacă obiectul este sigur invizibil, atunci se abandonează toate operațiile referitoare la acest obiect și se trece la obiectul următor.

Selecția obiectelor, adică identificarea obiectelor potențial vizibile pentru redarea imaginii acestora și eliminarea obiectelor sigur invizibile (*culling*), este o resursă puternică de creștere a eficienței de redare a scenelor virtuale. În scenele virtuale de dimensiuni mari, care conțin un număr mare de obiecte, în fiecare cadru de imagine sunt vizibile și deci trebuie să fie prelucrate un număr mult mai mic obiecte decât numărul total de obiecte ale scenei. De aceea, identificarea cât mai devreme posibil a obiectelor sigur invizibile și eliminarea lor conduce la reducerea substanțială a timpului de redare a scenei. În redarea scenelor complexe, acest procedeu se implementează mai eficient decât modul simplu descris anterior prin definirea volumului de delimitare a obiectelor.

#### 4.5.4 VOLUMUL DE DELIMITARE

Volumul de delimitare (*bounding box*) al unui obiect sau al unui grup de obiecte (ierarhie de obiecte) este un volum definit cât mai simplu, care include toate vârfurile obiectelor și are dimensiuni minime. Se folosesc ca volume de delimitare paralelipede dreptunghice sau sfere.

Fiind dat un obiect cu un volum de delimitare, testul de eliminare (*culling*) se efectuează asupra volumului de delimitare: se calculează intersecția codurilor de vizibilitate ale vârfurilor volumului de delimitare și, dacă este diferită de zero, atunci volumul de delimitare și, implicit, întregul obiect este invizibil și abandonat.

Această modalitate de reprezentare și prelucrare a obiectelor este aproape unanim adoptată în realitatea virtuală, datorită execuției eficiente a testului: decizia de eliminare a unui obiect se poate lua prin considerarea unui număr de opt vârfuri, în locul unui număr foarte mare de vârfuri cât are un obiect în mod obișnuit.

Volumul de delimitare al unui obiect se definește în sistemul de referință local (de modelare) al obiectului, ca un paralelipiped dreptunghic cu muchiile paralele cu axele de coordonate. Un astfel de paralelipiped se poate specifica prin coordonatele a două vârfuri opuse ( $x_{\min}$ ,  $y_{\min}$ ,  $z_{\min}$ ) și ( $x_{\max}$ ,  $y_{\max}$ ,  $z_{\max}$ ). Vârfurile lui sunt transformate din sistemul de referință local în alte sisteme de referință (universal, de observare, etc.), la fel ca și vârfurile obiectului. Testul de eliminare se poate efectua în sistemul de referință de observare sau în alt sistem de referință (sistemul normalizat).

În fig. 4.18 sunt reprezentate trei obiecte în poziții diferite față de volumul de vizualizare. Volumul de delimitare al primului obiect este complet exterior față de volumul de vizualizare și obiectul corespunzător (ceainic) este ignorat pentru punctul de observare dat. Cel de-al doilea obiect (icosaedru) este complet vizibil, volumul său de delimitare fiind inclus în volumul de vizualizare. Pentru cel de-al treilea obiect (sferă), volumul de delimitare intersectează volumul de vizualizare și trebuie să fie executată decuparea suprafețelor sale relativ la volumul de vizualizare.



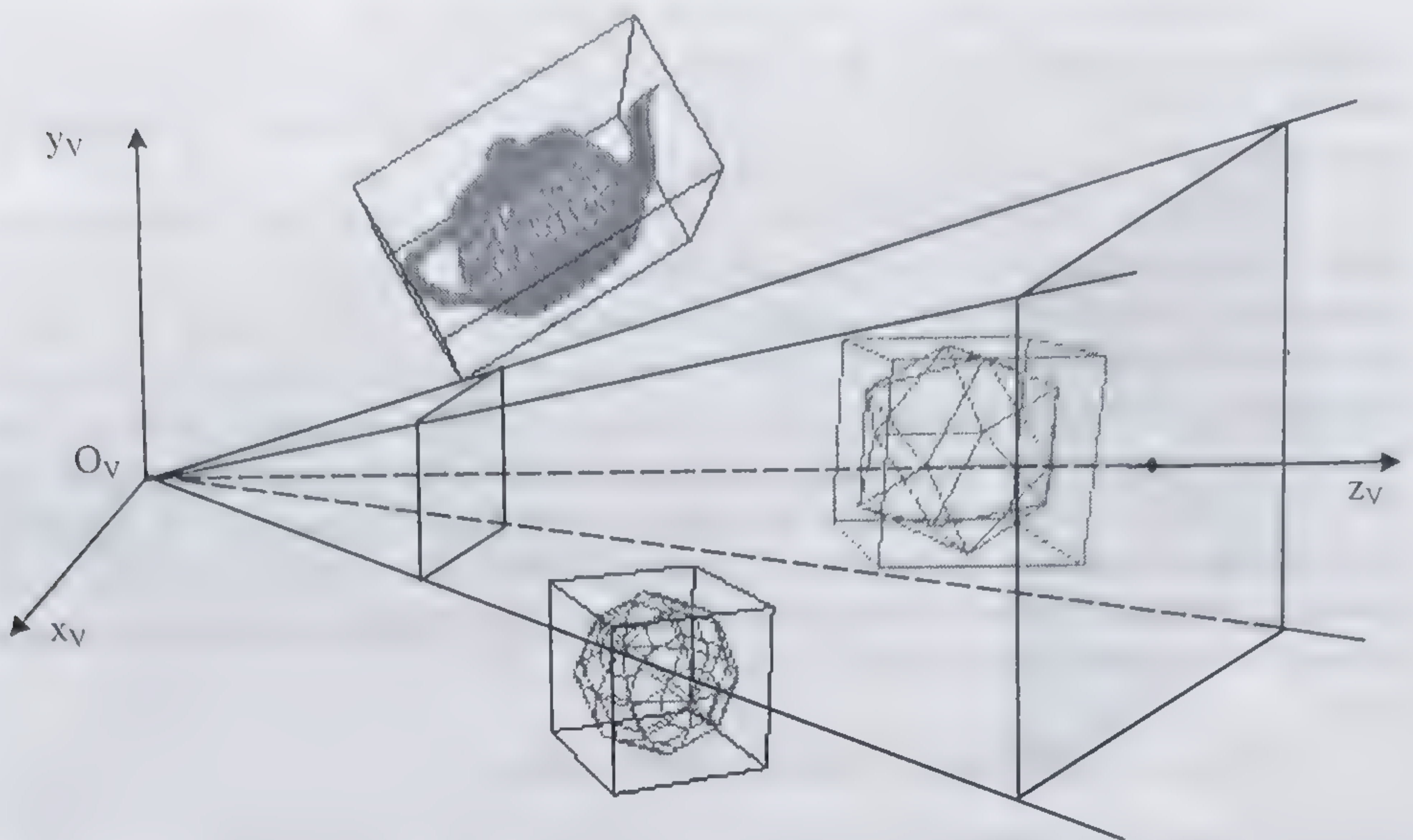


fig. 4.18 Eliminarea obiectelor pe baza volumului de delimitare (bounding box).

În scenele virtuale modelate ierarhic se construiesc volume de delimitare la fiecare nivel de ierarhie și obiectele sau grupurile de obiecte sunt selectate pe baza testului de eliminare efectuat asupra volumului de delimitare corespunzător.

#### 4.5.5 DETECȚIA COLIZIUNII

Deteția coliziunii este un aspect important în modelarea mediului virtual, prin faptul că simulează un aspect realist al mediului: obiectele reale nu pot trece unele prin altele; atunci când un obiect se atinge de un altul există un răspuns de coliziune: deformarea obiectelor, schimbarea traiectoriei de deplasare, explozia, etc. La interacțiunea dintre utilizator și obiectele virtuale se pot produce forțe, vibrații și alte reacții tactile care sunt prelucrate în sistem.

Deteția coliziunii se poate calcula aproximativ, folosind volumele de delimitare, sau exact, prin considerarea tuturor punctelor obiectelor. Dat fiind că volumul de delimitare depășește în multe puncte suprafața de frontieră a obiectului, calculul aproximativ al coliziunii introduce coliziuni false atunci când nu obiectele s-au atins, ci volumele lor de delimitare.

Deteția exactă a coliziunii obiectelor se efectuează după calculul coliziunii între volumele de delimitare. Dacă două obiecte au volumele de delimitare disjuncte, atunci ele sigur nu se ating. Dacă două obiecte au volume de delimitare care se ating sau interpătrund, atunci se calculează coliziunea exactă prin considerarea tuturor punctelor obiectelor.



## TRANSFORMAREA DE RASTRU

Transformarea de rastru (*rasterization*) este o conversie de la reprezentarea prin coordonatele vârfurilor a unui segment sau a unui poligon (primitivă geometrică), la reprezentarea prin mulțimea corespunzătoare de pixeli care se afișează pe display. Transformarea de rastru se mai numește conversie de baleiere (*scan conversion*), sau desenarea segmentelor, respectiv a poligoanelor (*line drawing*, *polygon drawing*) sau generare linie, respectiv poligon (*line generation*, *polygon generation*).

Transformarea de rastru implică trecerea de la spațiul bidimensional continuu (poarta de afișare definită în planul  $x_s = 0$  al sistemului de referință ecran 3D) la spațiul discret al imaginii (fig. 5.1)

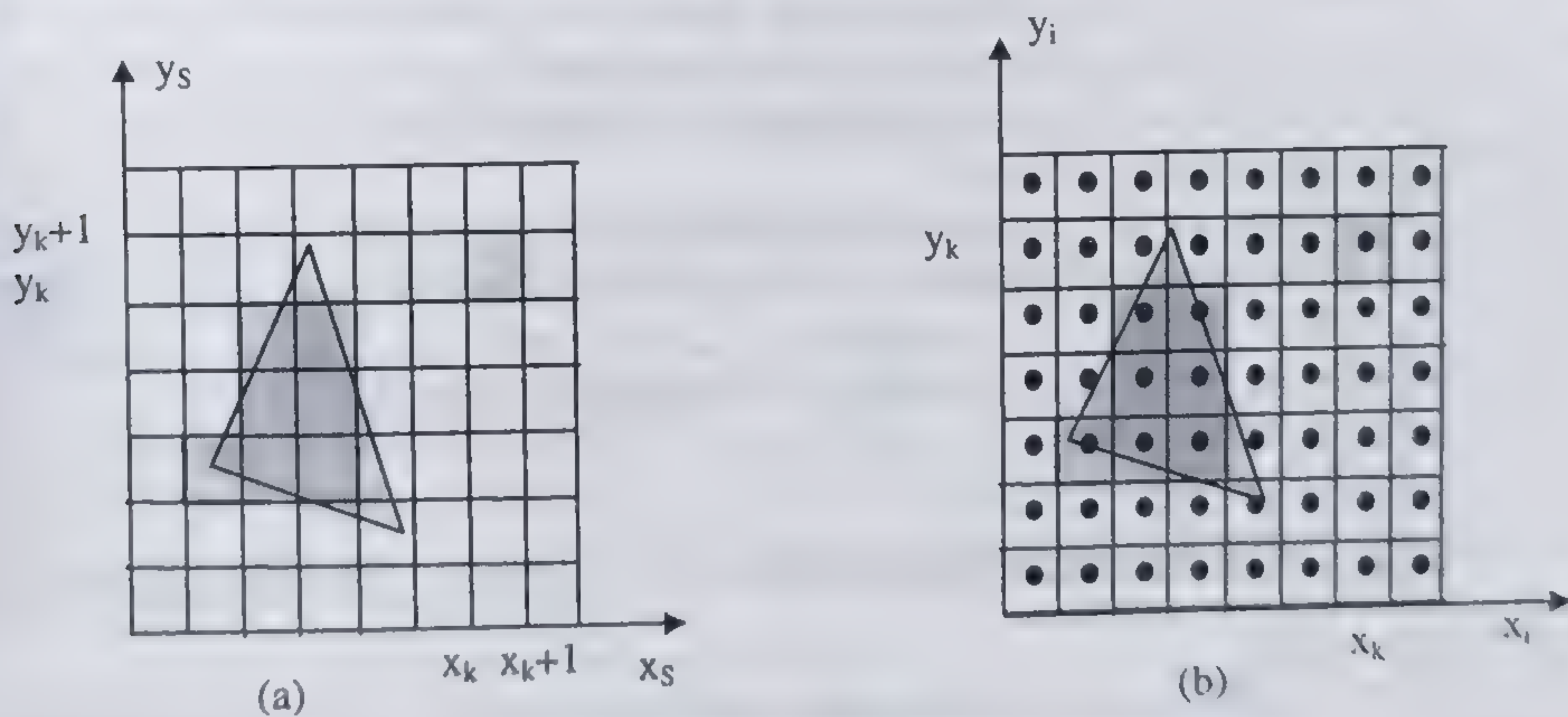


Fig. 5.1 (a) Spațiul bidimensional continuu al porții de afișare;  
(b) Spațiul bidimensional discret al imaginii.

Spațiul imagine este un tablou bidimensional de locații discrete numite pixeli, fiecare pixel având o adresă care corespunde poziției în imagine a acestuia. Dimensiunea imaginii este specificată prin numărul de pixeli pe orizontală ( $M$ ),



respectiv pe verticală (N), care pot fi afișați în zona de ecran atribuită porții de afișare. Dimensiunea maximă a imaginii depinde de rezoluția display-ului folosit.

Un pixel ocupă în poarta de afișare o zonă pătrată de dimensiune  $1 \times 1$  a cărei adresă este dată de coordonata colțului stânga-jos. De exemplu, pixelul de adresă  $(x_k, y_k)$  din spațiul imagine corespunde unui pătrat din poarta de afișare cu colțurile de coordonate  $(x_k, y_k)$   $(x_k+1, y_k)$   $(x_k+1, y_k+1)$   $(x_k, y_k+1)$ , unde  $x_k$  și  $y_k$  sunt numere întregi. În mod simplificat se consideră că imaginea pe ecran a pixelilor este formată din zone pătrate adiacente, dar, în realitate, imaginea pe ecran a unui pixel este o zonă circulară, iar aceste cercuri nu sunt adiacente ci se întrepătrund.

Un pixel din spațiul imagine se reprezintă prin culoarea lui, care se memorează într-un buffer de imagine. Adresa pixelului în spațiul imagine determină adresa în bufferul de imagine în care se memorează culoarea acestuia.

Din coordonatele  $x$  și  $y$  ale vârfurilor primitivelor geometrice în poarta de afișare se calculează ecuațiile segmentelor de dreaptă (laturile), iar mulțimea de pixeli generată în spațiul imagine trebuie să aproximeze cât mai bine segmentele de dreaptă sau interiorul poligoanelor delimitat prin laturile sale (fig.5.1).

Transformarea de rastu a segmentelor sau poligoanelor calculează adresa fiecărui pixel din mulțimea de pixeli prin care se aproximează primitiva geometrică dată și înscrie la adresa corespunzătoare din bufferul de imagine intensitatea de culoare a pixelului respectiv. Culoarea pixelului poate fi o culoare constantă, rezultată din atributul de culoare al primitivei geometrice sau din condiții de umbrire constantă (poligonală), și atunci această culoare se înscrie la toate adresele pixelilor care aparțin primitivei geometrice în bufferul de imagine. În cazul umbririi Gouraud, culoarea fiecărui pixel se calculează prin interpolare, concomitent cu calculul adresei pixelilor în transformarea de rastu.

Concomitent cu transformarea de rastu, se poate executa operația de eliminare a suprafețelor ascunse, folosind coordonata  $z_s$  (adâncimea) vârfurilor primitivelor geometrice calculate în sistemul de referință ecran 3D (algoritmul de tip buffer de adâncime (Z-buffer)).

În cele mai multe din sistemele grafice actuale, transformarea de rastu, umbrirea și eliminarea suprafețelor ascunse se execută intercorelat, într-o singură scanare (baleiere) a primitivei geometrice. La sistemele evolute se adaugă la acestea și operațiile de netezire a liniilor (anti-aliasing) și de texturare.

## 5.1 GENERAREA SEGMENTELOR DE DREAPTĂ

În orice bibliotecă grafică există o funcție care poate fi apelată pentru generarea unui segment de dreaptă. De exemplu funcția `line()` din biblioteca compilatorului Borland C, sau funcția `LineTo()` din interfața Win32API, trasează o linie în fereastra de afișare a programului.

Se consideră un segment de dreaptă AB dat prin capetele lui  $(x_A, y_A)$ ,  $(x_B, y_B)$  în sistemul de coordonate poarta de afișare (fig. 5.2). Ecuația dreptei care trece prin aceste puncte este  $y = ax + b$ , unde  $a = (y_B - y_A) / (x_B - x_A)$ ,  $b = y_A - a x_A$ .



În același sistem de referință, adresa unui pixel este dată de coordonatele colțului lui stânga-jos, deci pixelul de adresă  $(x_k, y_k)$  este un pătrat cu colțurile  $(x_k, y_k)$ ,  $(x_k+1, y_k)$ ,  $(x_k+1, y_k+1)$ ,  $(x_k, y_k+1)$ , unde  $x_k$  și  $y_k$  sunt numere întregi.

Dacă  $x_1 \leq x_2$ , atunci se calculează secvența de puncte pe dreaptă:

$$(x_1 = x_A, y_1 = ax_1 + b),$$

$$(x_2 = x_1 + 1, y_2 = ax_2 + b), \dots, (x_i, y_i = ax_i + b), \dots$$

Fiecărui punct pe dreaptă  $i$  se asociază un pixel a cărui adresă în sistemul de referință imagine se aproximează prin trunchierea sau rotunjirea coordonatelor punctului în sistemul de referință de afișare:

$$x_i' = (\text{int})x_i \text{ sau } x_i' = (\text{int})(x_i + 0.5)$$

$$y_i' = (\text{int})y_i \text{ sau } y_i' = (\text{int})(y_i + 0.5)$$

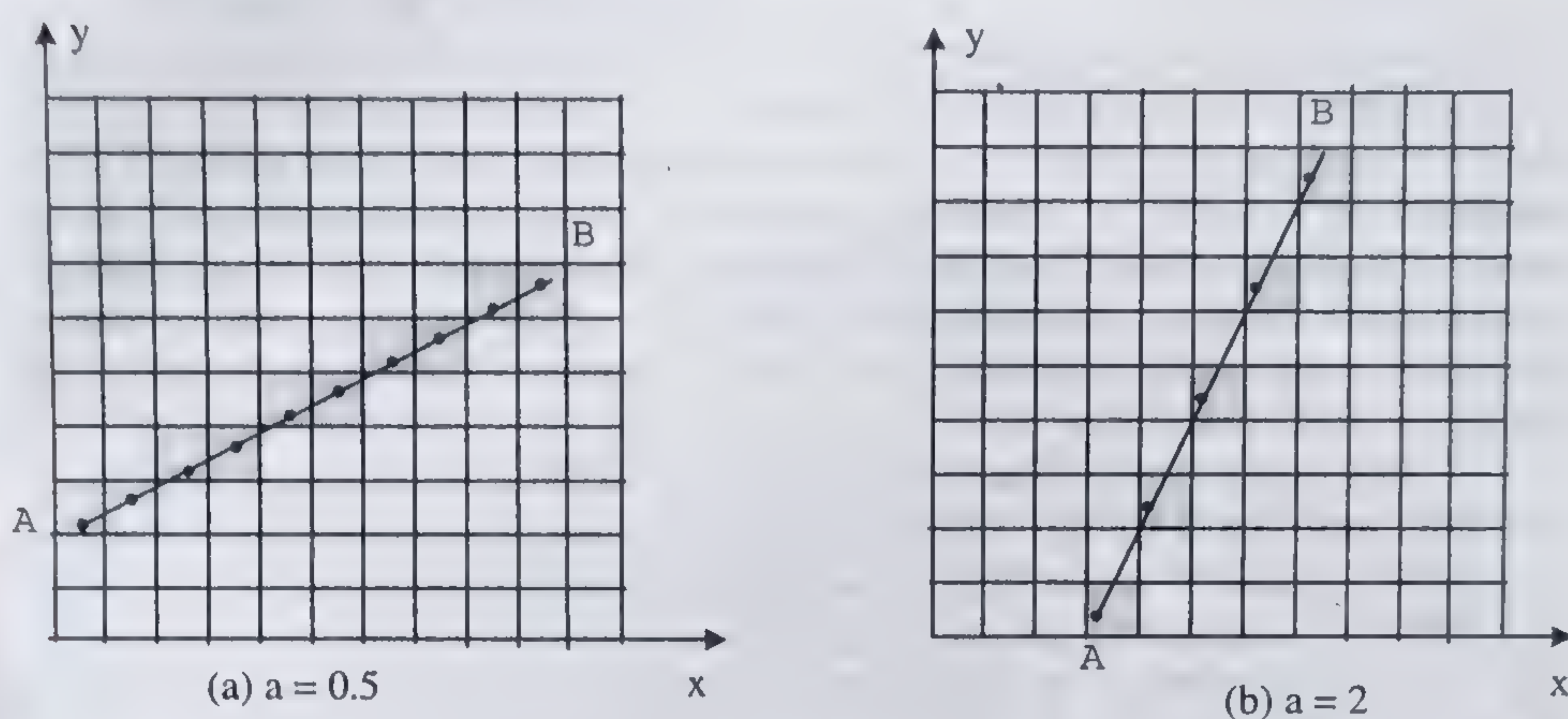


Fig. 5.2 Generarea segmentelor de dreaptă prin aplicarea directă a relațiilor de calcul.

În acest mod se obține transformarea în spațiul imagine a unui segment de dreaptă, reprezentat prin mulțimea pixelilor care aproximează segmentul de dreaptă dat.

Aplicarea directă a formulei de calcul al punctelor de pe dreaptă are dezavantajul că necesită mai multe operații (adunare, înmulțire) în numere flotante pentru fiecare pixel, operații care au timp de execuție ridicat. Un alt dezavantaj al acestei metode este că generează segmente de dreaptă întrerupte dacă  $a > 1$  (fig. 5.2(b)).

Înlăturarea acestor inconveniente se face prin utilizarea algoritmilor incrementali, în care se evită repetarea operațiilor de înmulțire în virgulă flotantă, folosind trecerea prin incrementare de la un punct la următorul. Pentru obținerea segmentelor de dreaptă fără întrerupri se tratează separat cazurile  $\text{abs}(a) \leq 1$  și  $\text{abs}(a) > 1$ .

Dacă  $\text{abs}(a) \leq 1$  și  $x_A \leq x_B$ , se începe generarea din punctul  $(x_1 = x_A, y_1 = y_A)$  și la fiecare pas se incrementează  $x$  cu 1, deci se iau punctele de pe dreaptă care au:  $x_1 = x_A, x_2 = x_1 + 1, \dots, x_i, x_{i+1} = x_i + 1, \dots$ . Dat fiind că  $y_i = ax_i + b$ , rezultă că



$y_{i+1} = ax_{i+1} + b = a(x_i + 1) + b = y_i + a$ . Deci, pentru trecerea de la un punct la următorul,  $x$  se incrementează cu 1, iar  $y$  se incrementează cu valoarea pantei. Punctului de pe dreaptă de coordonate  $(x_i, y_i)$ , îi corespunde un pixel a cărui adresă în spațiul imagine se calculează prin trunchiere sau rotunjire.

Dacă  $\text{abs}(a) \leq 1$  și  $x_A > x_B$ , se începe generarea din punctul  $(x_1 = x_B, y_1 = y_B)$  și restul calculelor se continuă la fel ca în cazul precedent.

Dacă  $\text{abs}(a) > 1$  și  $x_A \leq x_B$ , se începe generarea segmentului din punctul  $(x_1 = x_A, y_1 = y_A)$  și la fiecare pas se incrementează  $y$  cu 1, deci se calculează punctele care au:  $y_1 = y_A, y_2 = y_1 + 1, \dots, y_i, y_{i+1} = y_i + 1, \dots$ . Deoarece  $x_i = (y_i - b)/a$ , rezultă că  $x_{i+1} = (y_{i+1} - b)/a = x_i + 1/a$ . Deci, pentru trecerea de la un punct la următorul,  $y$  se incrementează cu 1, iar  $x$  se incrementează cu inversul pantei.

Dacă  $\text{abs}(a) > 1$  și  $x_A > x_B$ , se începe generarea din punctul  $(x_1 = x_B, y_1 = y_B)$  și restul calculelor se continuă la fel ca în cazul precedent.

Algoritmul incremental de generare a primitivelor geometrice are mai multe variante de implementare, în funcție de situația în care este aplicat. Pentru segmente de dreaptă, se adaptează algoritmul incremental astfel încât să se genereze segmente fără întreruperi, folosind numai operații cu numere întregi. Această variantă este cunoscută sub numele de algoritmul Bresenham. Pentru generarea poligoanelor, varianta de algoritm incremental este algoritmul de baleiere pe linii (*scan-line*).

Algoritmul Bresenham se bazează pe metoda incrementală, dar este optimizat astfel încât folosește numai operații cu numere întregi. Datorită acestei proprietăți, el poate fi implementat hardware și, în majoritatea cazurilor, operațiile de generare a segmentelor de dreaptă din acceleratoarele grafice se bazează pe o implementare hardware a algoritmului Bresenham. Descrierea detaliată a algoritmului Bresenham se poate găsi în bibliografie [Bre65], [Mold96].

Indiferent de algoritmul de conversie folosit pentru generarea liniilor, trecerea de la spațiul bidimensional continuu, în care poziția punctelor este reprezentată prin numere reale, la reprezentarea în spațiul discret al imaginii este o aproximare prin eșantionare, care produce erori de reprezentare. Din cauza acestei aproximări, orice linie dreaptă (cu excepția celor orizontale sau verticale) apare zimțată (*jagged*), ca o "scară" de segmente de dreaptă succesive, așa cum se poate observa în orice imagine generată pe un display sau reprodusă pe imprimantă, inclusiv în textul de față. Imaginea de scară este cu atât mai pregnantă cu cât rezoluția display-ului (numărul de pixeli pe orizontală și pe verticală) este mai mică. Dacă rezoluția crește, aspectul nedorit de scară este atenuat, dar nu este eliminat. Netezirea liniilor se poate obține prin diferite metode de filtrare, metode cunoscute sub numele de filtrare anti-aliasing.

## 5.2 GENERAREA POLIGOANELOR

Metoda cea mai frecvent folosită pentru generarea mulțimii pixelilor prin care se aproximează o suprafață poligonală este metoda de baleiere pe linii a



poligonului (algoritmul *scan-line*). Principiul de bază al acestei metode pentru un poligon convex este prezentat în fig. 5.3.

Se consideră poligonul  $A'B'C'D'E'$ , care reprezintă proiecția în planul  $z_s = 0$  al sistemului de referință ecran 3D, a unei suprafețe ABCDE din spațiul tridimensional universal. Poligonul este baleiat cu o linie orizontală care ia valorile succesive  $y_1, y_2, \dots, y_i, y_{i+1}, \dots$ , unde  $y_1$  este valoarea coordonatei  $y$  minimă a vârfurilor poligonului, iar  $y_{i+1} = y_i + 1$ . Pentru fiecare poziție  $y_i$  a liniei de baleiere, se calculează intersecțiile  $x_{i,1}, x_{i,2}$  cu două din laturile poligonului. Pixelii a căror adrese sunt cuprinse în intervalul  $(x_{i,1}, y_i), (x_{i,2}, y_i)$ , aparțin poligonului și sunt prelucrați în mod corespunzător. Pentru algoritmul simplu de generare a poligoanelor, culoarea poligonului (cunoscută ca atribut a feței corespunzătoare a obiectului) este înscrisă în bufferul de imagine la toate adresele cuprinse în intervalul respectiv. În algoritmi în care se combină generarea cu eliminarea suprafețelor ascunse sau umbrire se prelucrează toți pixelii corespunzători acestui interval.

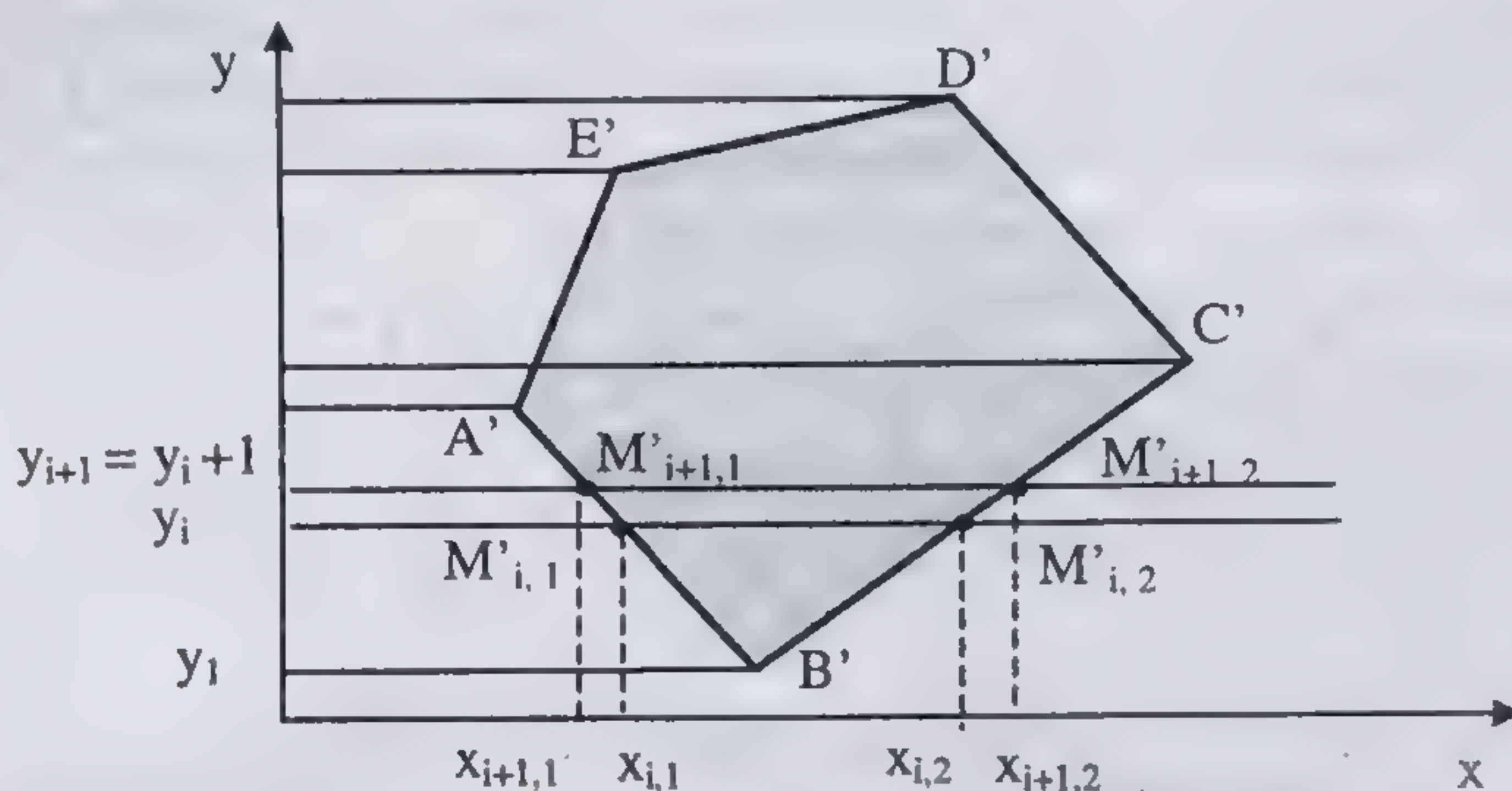


Fig. 5.3 Principiul baleierii pe linii a poligoanelor (*scan-line*).

Fie următoarele ecuații ale laturilor stânga și dreapta cu care se intersectează dreapta  $y = y_i$ :

$$\begin{aligned} x &= m_1 y + n_1 \\ x &= m_2 y + n_2 \end{aligned} \quad (5.1)$$

Pentru situația din fig. 5.3, intersecția dreptei  $y = y_i$  are loc cu laturile  $A'B'$  și  $B'C'$ , pentru care coeficienții din ecuațiile de mai sus au valorile:

$$\begin{aligned} m_1 &= (x_A - x_B) / (y_A - y_B), \quad n_1 = x_B - m_1 y_B \\ m_2 &= (x_C - x_B) / (y_C - y_B), \quad n_2 = x_C - m_2 y_C \end{aligned} \quad (5.2)$$

Intersecțiile corespunzătoare sunt:  $x_{i,1} = m_1 y_i + n_1$ ,  $x_{i,2} = m_2 y_i + n_2$ . Coordonatele intersecțiilor liniei următoare de baleiere  $y = y_{i+1}$ , unde  $y_{i+1} = y_i + 1$ , se pot calcula din coordonatele precedente folosind coeficienții  $m_1$  și  $m_2$  ai laturilor intersectate:

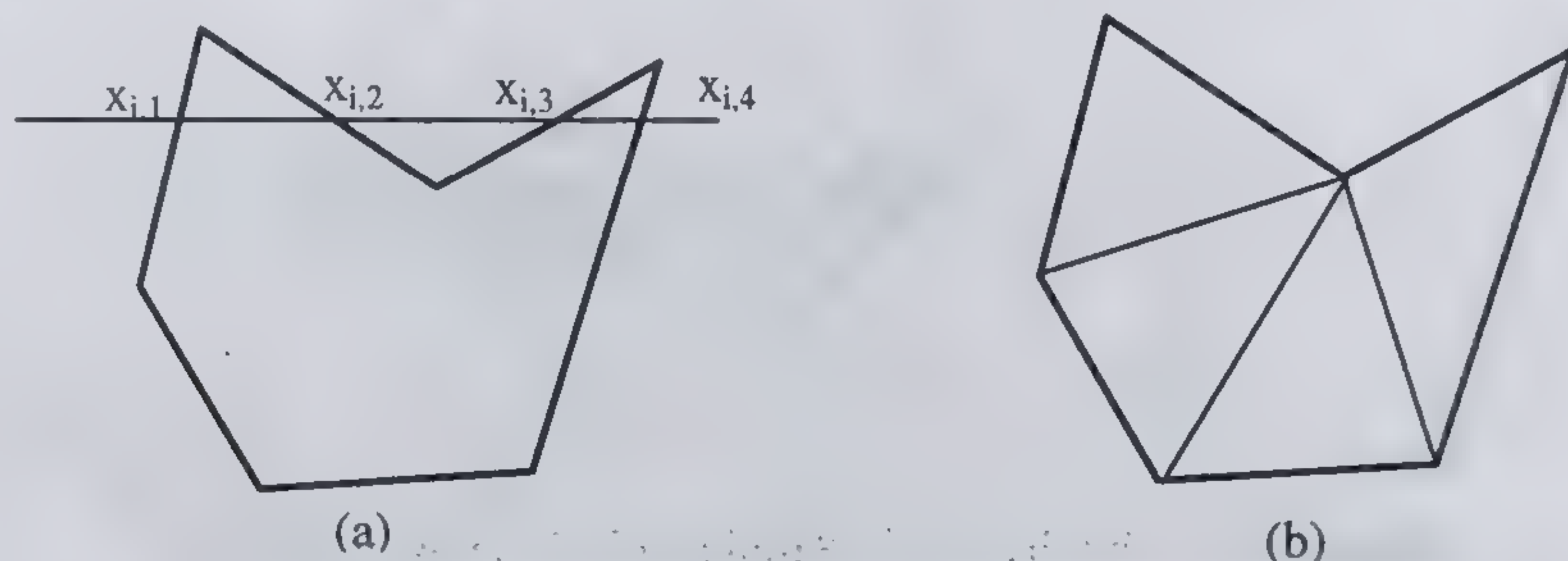
$$\begin{aligned} x_{i+1,1} &= x_{i,1} + m_1 \\ x_{i+1,2} &= x_{i,2} + m_2 \end{aligned} \quad (5.3)$$



Secvența de pixeli ai căror adrese sunt cuprinse în intervalul  $(x_{i+1,1}, y_{i+1})$ ,  $(x_{i+1,2}, y_{i+1})$  aparțin poligonului pe linia de baleiere  $y = y_{i+1}$  și sunt tratați corespunzător.

Acesta este principiul de bază al baleierii pe linii (*scan-line*). Cazurile particulare, când se schimbă latura cu care are loc intersecția (la colțurile poligonului), sau când există laturi orizontale, se tratează separat.

Algoritmul de baleiere pe linie se poate implementa și pentru poligoane concave. În acest caz, o linie de baleiere poate intersecta mai multe perechi de laturi, rezultând mai multe segmente de interecție și deci mai multe secvențe de pixeli care aparțin poligonului (fig. 5.4(a)). În general, însă, poligoanele concave sau cu găuri se tratează pe mai multe nivele. Acceleratoarele grafice implementează hardware automate de generare a unor suprafețe convexe simple (triunghiuri sau trapeze). Descompunerea unui poligon convex în mai multe triunghiuri este o operație simplă care, de cele mai multe ori, este preluată de biblioteca grafică, care conține funcții de generare a poligoanelor convexe cu un număr oarecare de laturi. Descompunerea poligoanelor concave în poligoane convexe este tratată la nivelul programului de aplicație. Această ierarhizare în prelucrarea poligoanelor este precizată în capitolul 6 referitor la biblioteca OpenGL.



**Fig. 5.4** Generarea poligoanelor concave:

- (a) generare directă prin baleiere pe linii;
- (b) transformare în mulțime de triunghiuri (triangularizare).

Ca și în cazul liniilor, muchiile poligoanelor generate prezintă aspectul de scară de segmente, datorită trecerii la spațiul discret al imaginii. Algoritmii de filtrare pentru netezirea muchiilor poligonului (algoritmi de anti-aliasing) se execută, în general, concomitent cu algoritmul de generare scan-line a poligonului.

### 5.3 ELIMINAREA SUPRAFETELOR ASCUNSE

Reprezentarea pe display a tuturor suprafețelor obiectelor proiectate în fereastra de vizualizare produce o imagine nerealistă, deoarece este posibil ca o suprafață care se află în spatele altei suprafețe să fie vizibilă pe ecran. Această situație provine din transformarea de proiecție din spațiul tridimensional în spațiul



bidimensional (planul de vizualizare). În general, orice proiecție înseamnă o reducere a dimensionalității spațiului de reprezentare și, prin acesta, se pierde o parte din proprietățile formelor. Datorită proiecției, suprafețe distincte, aflate la distanțe diferite în spațiu, sunt proiectate în aceeași regiune pe display și sunt reprezentate suprapus. Pentru obținerea unor imagini corecte, care să reflecte relația spațială dintre obiecte, se prelucraază suplimentar suprafețele proiectate ale obiectelor.

Prelucrarea suprafețelor din punct de vedere al poziției relative între ele, prelucrare cunoscută sub numele de eliminarea suprafețelor ascunse (*hidden surface removal*) se poate efectua în două moduri:

- (a) Eliminarea suprafețelor ascunse în spațiul obiect, prin compararea adâncimii relative a suprafețelor obiectelor în sistemul de referință de observare sau în sistemul de referință normalizat.
- (b) Eliminarea suprafețelor ascunse în spațiul imagine, prin compararea adâncimii pixelilor imaginii rezultați din generarea (conversia de rastru) a tuturor suprafețelor obiectelor, prin algoritmul cunoscut sub numele de algoritm de tip buffer de adâncime (*depth buffer, Z-buffer*).

În momentul actual, eliminarea suprafețelor ascunse prin algoritmul Z-buffer este aproape universal folosită în grafică, datorită calităților remarcabile ale acestei abordări, care permite implementări hardware eficiente și este independentă de modul de reprezentare a obiectelor scenei. Alți algoritmi de eliminare a suprafețelor ascunse se mai folosesc doar în aplicații grafice foarte specializate, care nu beneficiază de mecanismul de Z-buffer.

### 5.3.1 COMPARAREA ADÂNCIMILOR

Pentru eliminarea suprafețelor ascunse se compară adâncimea (distanța față de punctul de observare) tuturor suprafețelor sau a pixelilor rezultați prin conversia de rastru.

Un punct  $P$  este ascuns de o suprafață  $S$ , dacă adâncimea punctului este mai mare decât adâncimea intersecției  $I$  a dreapta care unește punctul  $P$  cu centrul de proiecție (se consideră centrul de proiecție identic cu originea sistemului de referință de observare  $O_v$ , fig. 5.5(a)), deci dacă  $O_v P > O_v I$ .

O suprafață  $A$  este ascunsă de altă suprafață  $S$  dacă toate punctele suprafeței  $A$  sunt ascunse de suprafața  $S$  (fig. 5.5(b)). Dacă o suprafață ascunde  $S$  suprafața  $A$ , atunci se consideră că adâncimea suprafeței  $S$  este mai mică decât cea a suprafeței ascunse  $A$ . Compararea adâncimilor a două puncte în sistemul de referință de observare este destul de complicată, deoarece comparația are sens numai dacă cele două puncte se află pe aceeași linie de proiecție (fig. 5.5).

Comparația adâncimilor este mai simplă în sistemul de referință ecran 3D. În acest sistem,  $z_S = z_N$ , iar  $z_N$  este definit de relația (4.10) astfel:

$$z_S = z_N = \frac{f}{f - d} \left( 1 - \frac{d}{z_v} \right)$$



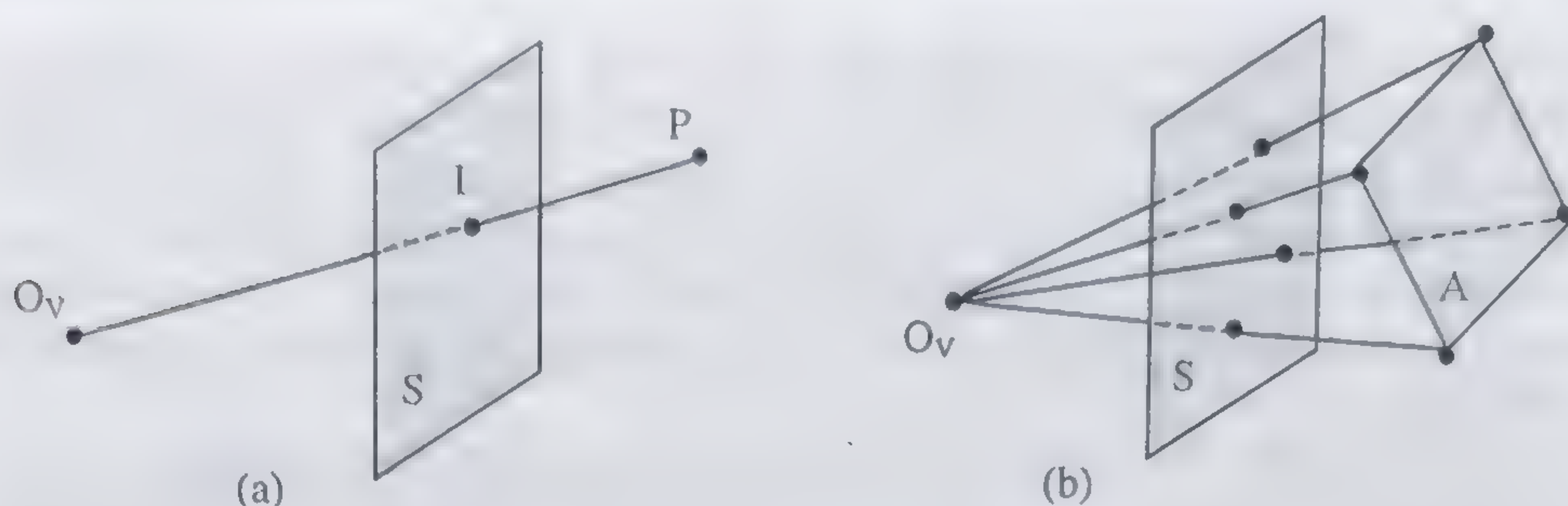


Fig. 5.5 (a) Suprafața  $S$  ascunde punctul  $P$ .  
(b) Suprafața  $S$  ascunde suprafața  $A$ .

Această relație transformă, de exemplu, un cub în sistemul de referință de observare, într-un trunchi de piramidă în sistemul ecran 3D (fig. 5.6).

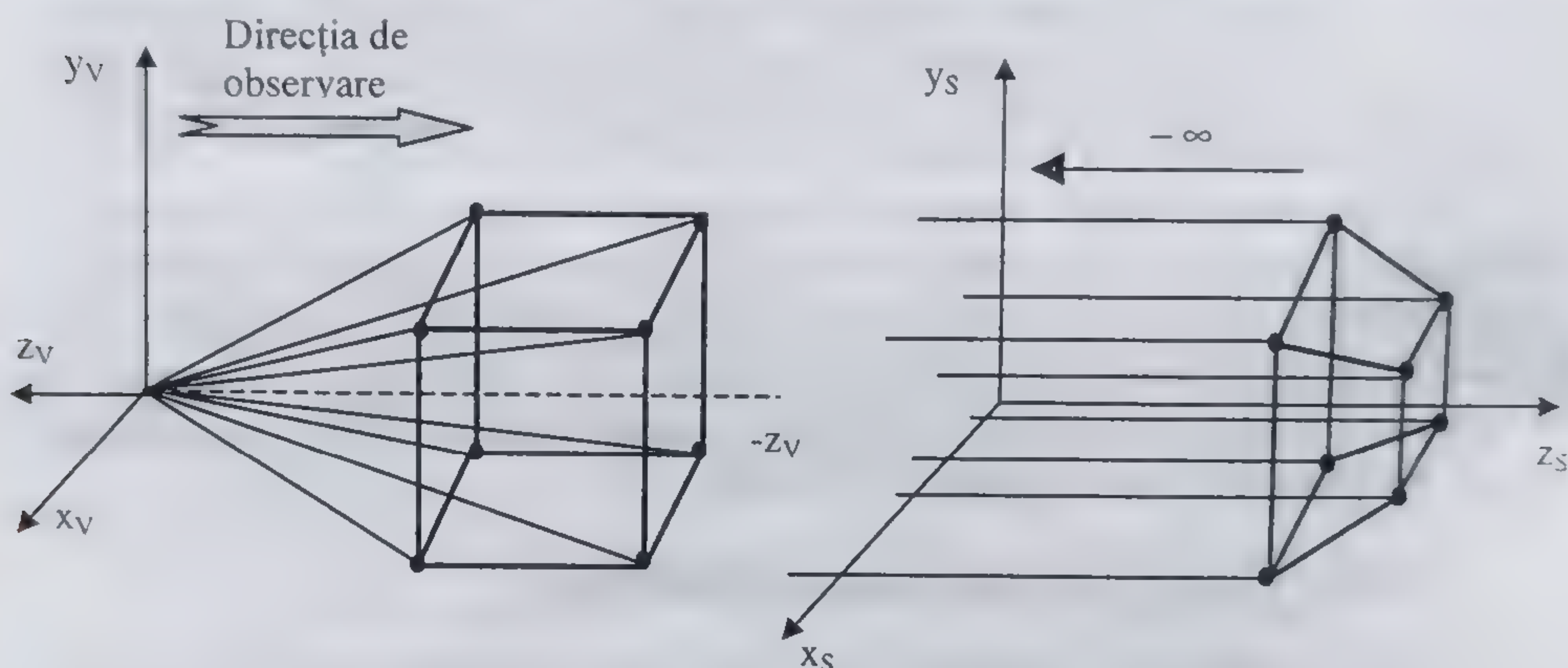


Fig. 5.6 Transformarea unui cub și a liniilor de proiecție din sistemul de referință de observare în sistemul de referință ecran 3D.

Proiecția perspectivă din sistemul de referință de observare se transformă în proiecție paralelă în sistemul de referință ecran 3D. Deoarece  $z_v = 0$  se transformă în  $z_s = -\infty$ , liniile de proiecție în sistemul ecran 3D sunt linii paralele îndreptate către  $-\infty$ . Adâncimea unui punct  $P_1(x_1, y_1, z_1)$  în sistemul de referință de observare, care este dată de distanța  $O_v P_1 = \sqrt{x_1^2 + y_1^2 + z_1^2}$ , se transformă în coordonata  $z_s$  a punctului în sistemul de referință ecran 3D.

În sistemul de referință de observare, testul de adâncime pentru a determina dacă un punct  $P_1$  este mai aproape de centrul de proiecție (care este, în general, același cu punctul de observare) decât punctul  $P_2$ , se poate efectua numai dacă cele două puncte se află pe aceeași linie de proiecție. În sistemul ecran 3D, toate punctele care au aceleași coordonate  $(x_s, y_s)$  în planul de proiecție se află pe același proiector, o dreaptă paralelă cu axa  $z_s$ , deci se poate efectua testul de adâncime prin comparația coordonatelor  $z_s$  ale punctelor.



Se poate trage concluzia că operațiile care implică testarea sau compararea adâncimilor se execută mai simplu în sistemul de referință ecran 3D. Acest lucru este utilizat atât în algoritmi de eliminare a suprafețelor ascunse în spațiul obiect, cât și în algoritmi în spațiul imagine.

### 5.3.2 ELIMINAREA SUPRAFETELOR ASCUNSE ÎN SPAȚIUL OBIECT

Unul din cei mai cunoscuți algoritmi de eliminare a suprafețelor ascunse în spațiul obiect este algoritmul propus de Newell, cunoscut și sub numele de *algoritmul pictorului*, datorită analogiei cu modul în care un pictor creează un tablou [New72]. În acest algoritm, se sortează suprafețele în ordinea descrescătoare a adâncimii și se redau în această ordine pe display. Eliminarea suprafețelor ascunse se obține prin suprapunerea noii suprafețe (mai apropiată) peste suprafețele mai depărtate, deja desenate. Lista ordonată de suprafețe (numită listă de display) se creează în sistemul de referință de observare sau în sistemul de referință ecran 3D, după operația de decupare.

Ordonarea suprafețelor după adâncime este o operație complicată, mai ales dacă două suprafețe se intersectează între ele, situație în care nu se poate decide care suprafață este mai apropiată, decât prin divizarea uneia din ele de-a lungul liniei de intersecție.

Un alt algoritm de eliminare a suprafețelor ascunse în spațiul obiect este algoritmul cu subdivizarea recursivă a poligoanelor (algoritmul Weiler-Atherton, [Wei77]). Spre deosebire de algoritmul pictorului, în care intersectarea suprafețelor proiectate se efectuează prin supraînscrierea suprafețelor mai apropiate peste cele mai depărtate, în algoritmul Weiler-Atherton se execută o operație de decupare în sistemul de referință normalizat între toate suprafețele obiectelor și se rețin numai suprafețele "învingătoare", adică cele mai apropiate. Lista acestora (lista de display) nu mai conține suprafețe care se pot suprapune pe ecran și nu mai trebuie să fie ordonată. Ca și în algoritmul pictorului, operațiile de creare a listei de display necesită un timp de execuție foarte mare, care depinde de numărul de suprafețe ce se prelucrează. Acest lucru limitează drastic posibilitatea de utilizare a algoritmilor în spațiul obiect.

### 5.3.3 ELIMINAREA SUPRAFETELOR ASCUNSE ÎN SPAȚIUL IMAGINE: ALGORITMUL Z-BUFFER

Dezvoltat de Catmull în anul 1975 [Cat75], algoritmul Z-buffer introduce o metodă simplă de eliminare a suprafețelor ascunse, în care se efectuează comparația între adâncimea  $z_s$  a fiecărui pixel  $(x, y)$  al poligoanelor generate și se selectează pixelul cu adâncimea minimă (cel mai apropiat).

Această selecție se implementează cel mai convenabil prin folosirea unui buffer de adâncime (Z-buffer), care memorează adâncimea tuturor pixelilor generați în spațiul de imagine, a căror culoare este memorată în bufferul de imagine. În orice moment, la fiecare adresă  $(x, y)$  în Z-buffer, este memorată



Se poate trage concluzia că operațiile care implică testarea sau compararea adâncimilor se execută mai simplu în sistemul de referință ecran 3D. Acest lucru este utilizat atât în algoritmi de eliminare a suprafețelor ascunse în spațiul obiect, cât și în algoritmi în spațiul imagine.

### 5.3.2 ELIMINAREA SUPRAFETELOR ASCUNSE ÎN SPAȚIUL OBIECT

Unul din cei mai cunoscuți algoritmi de eliminare a suprafețelor ascunse în spațiul obiect este algoritmul propus de Newell, cunoscut și sub numele de *algoritmul pictorului*, datorită analogiei cu modul în care un pictor creează un tablou [New72]. În acest algoritm, se sortează suprafețele în ordinea descrescătoare a adâncimii și se redau în această ordine pe display. Eliminarea suprafețelor ascunse se obține prin suprapunerea noii suprafețe (mai apropiată) peste suprafețele mai depărtate, deja desenate. Lista ordonată de suprafețe (numită listă de display) se creează în sistemul de referință de observare sau în sistemul de referință ecran 3D, după operația de decupare.

Ordonarea suprafețelor după adâncime este o operație complicată, mai ales dacă două suprafețe se intersectează între ele, situație în care nu se poate decide care suprafață este mai apropiată, decât prin divizarea uneia din ele de-a lungul liniei de intersecție.

Un alt algoritm de eliminare a suprafețelor ascunse în spațiul obiect este algoritmul cu subdivizarea recursivă a poligoanelor (algoritmul Weiler-Atherton, [Wei77]). Spre deosebire de algoritmul pictorului, în care intersectarea suprafețelor proiectate se efectuează prin supraînscrisura suprafețelor mai apropiate peste cele mai depărtate, în algoritmul Weiler-Atherton se execută o operație de decupare în sistemul de referință normalizat între toate suprafețele obiectelor și se rețin numai suprafețele "învingătoare", adică cele mai apropiate. Lista acestora (lista de display) nu mai conține suprafețe care se pot suprapune pe ecran și nu mai trebuie să fie ordonată. Ca și în algoritmul pictorului, operațiile de creare a listei de display necesită un timp de execuție foarte mare, care depinde de numărul de suprafețe ce se prelucrează. Acest lucru limitează drastic posibilitatea de utilizare a algoritmilor în spațiul obiect.

### 5.3.3 ELIMINAREA SUPRAFETELOR ASCUNSE ÎN SPAȚIUL IMAGE: ALGORITMUL Z-BUFFER

Dezvoltat de Catmull în anul 1975 [Cat75], algoritmul Z-buffer introduce o metodă simplă de eliminare a suprafețelor ascunse, în care se efectuează comparația între adâncimea  $z_s$  a fiecărui pixel  $(x, y)$  al poligoanelor generate și se selectează pixelul cu adâncimea minimă (cel mai apropiat).

Această selecție se implementează cel mai convenabil prin folosirea unui buffer de adâncime (Z-buffer), care memorează adâncimea tuturor pixelilor generați în spațiul de imagine, a căror culoare este memorată în bufferul de imagine. În orice moment, la fiecare adresă  $(x, y)$  în Z-buffer, este memorată



adâncimea celui mai apropiat pixel de adresă  $(x, y)$  rezultat în transformarea de rastru a poligoanelor. Adâncimea unui nou pixel generat este comparată cu adâncimea pixelului cu aceeași adresă din Z-buffer și noul pixel înlocuiește pixelul anterior numai dacă adâncimea lui este mai mică. Înlocuirea pixelului anterior cu un pixel nou generat se realizează prin înscrierea culorii noului pixel în bufferul de imagine la adresa  $(x, y)$  și înscrierea adâncimii lui la aceeași adresă  $(x, y)$  în Z-buffer.

Comparația între adâncimile punctelor în sistemul de referință ecran 3D are avantajul că poate fi efectuată prin comparația coordonatelor  $z_s$ , dat fiind că toate punctele cu aceleași coordonate  $(x, y)$  se află pe aceeași linie de proiecție paralelă cu axa  $z_s$  (fig. 5.8). Dar această simplitate de calcul are, totuși, un inconvenient, și anume acela că adâncimile din sistemul de referință de observare nu se transformă liniar în sistemul de referință ecran 3D. Intervale egale în  $z_v$  nu se transformă în intervale egale în  $z_s$ , ceea ce se poate observa și din relațiile de transformare (4.10). Cu cât  $z_v$  se apropie de planul de vizibilitate depărtat (*far*),  $z_s$  se apropie mai rapid de 1. Obiectele sunt comprimate în funcție de coordonata lor  $z_v$ , și acest lucru are consecințe asupra preciziei în comparația adâncimilor.

Algoritmul Z-buffer se execută concomitent cu transformarea de rastru a poligoanelor. Fiecare poligon este descris prin lista vârfurilor sale reprezentate în sistemul de referință ecran 3D. Adâncimea pixelilor poligonului se calculează prin interpolare pornind de la adâncimea vârfurilor acestuia.

Interpolarea pentru calculul adâncimii pixelilor se efectuează în două etape: o interpolare de-a lungul laturilor poligonului, prin care se obține adâncimea pixelilor de pe conturul poligonului, și o interpolare de-a lungul fiecărei linii de baleiere, prin care se obține adâncimea tuturor pixelilor interiori. Algoritmul de generare a poligoanelor prin baleiere pe linii (scan-line) se transformă astfel într-un algoritm combinat de baleiere și Z-buffer (*scan-line Z-buffer*).

Se reia algoritmul de generare a poligoanelor prin baleiere pe linii din subcapitolul 5.2. Prin proiecția suprafeței ABCDE din spațiu în planul  $x_s = 0$  se obține poligonul A''B''C''D''E'' (fig. 5.7). Liniile de baleiere ale acestui poligon sunt drepte de ecuații  $y = y_1, y = y_2, \dots, y = y_i, y = y_{i+1}, \dots$ . Dreapta  $y = y_i$  interesează laturile A''B'' și B''C'' în punctele M''<sub>i,1</sub> ( $y_i, z_{i,1}$ ) și M''<sub>i,2</sub> ( $y_i, z_{i,2}$ ).

Ecuațiile laturilor A''B'' și B''C'' se pot scrie similar cu relațiile (5.1):

$$\begin{aligned} z &= m_{1z}y + n_{1z} \\ z &= m_{2z}y + n_{2z} \end{aligned} \quad (5.4)$$

$$\text{unde: } \begin{aligned} m_{1z} &= (z_A - z_B) / (y_A - y_B), \quad n_{1z} = y_B - m_{1z} z_B \\ m_{2z} &= (z_C - z_B) / (y_C - y_B), \quad n_{2z} = y_C - m_{1z} z_C \end{aligned} \quad (5.5)$$

Intersecțiile dreptelor A''B'' și B''C'' cu dreapta  $y = y_i$  au coordonatele  $z_{i,1} = m_{1z}y_i + n_{1z}$  și  $z_{i,2} = m_{2z}y_i + n_{2z}$ . Coordonatele punctelor de intersecție ale dreptei  $y = y_{i+1} = y_i + 1$ , se pot calcula din coordonatele intersecțiilor precedente folosind coeficienții  $m_{1z}$  și  $m_{2z}$  ai laturilor intersectate:

$$\begin{aligned} z_{i+1,1} &= z_{i,1} + m_{1z} \\ z_{i+1,2} &= z_{i,2} + m_{2z} \end{aligned} \quad (5.6)$$



Pentru fiecare linie de baleiere se calculează coordonatele intersecțiilor cu laturile poligonului prin incrementarea valorilor precedente cu coeficienții dați de relațiile (5.5).

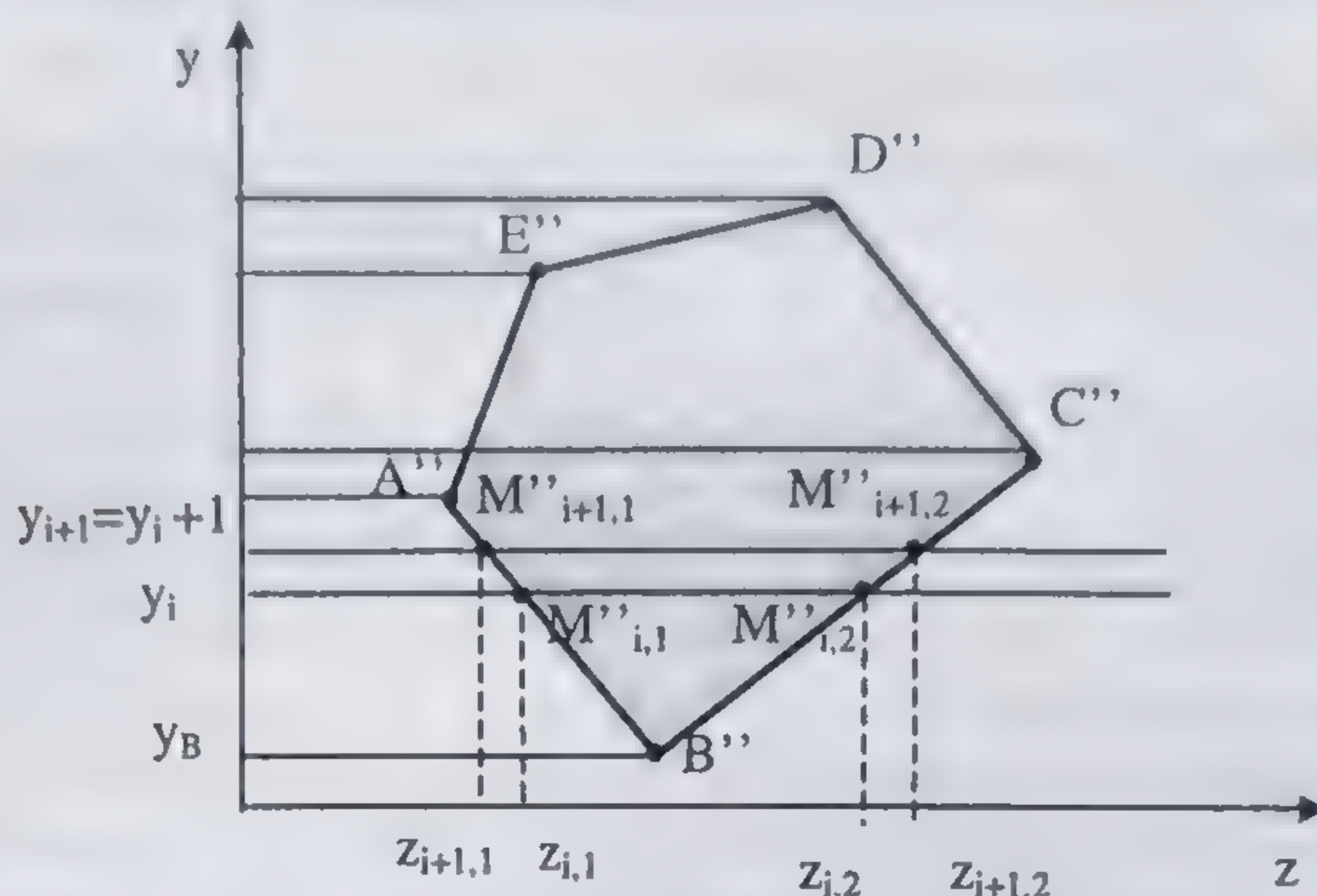


Fig. 5.7 Baleierea poligonului în planul yz.

Cele două operații de baleiere, pentru calculul adresei și al adâncimii pixelilor de pe conturul unei suprafețe ABCDE, se execută concomitent prin intersecția suprafeței cu planele de ecuații  $y = y_1, y = y_2, \dots, y_i, y_{i+1}, \dots$ . Laturile AB și BC sunt intersectate de planul  $y = y_i$  în punctele  $M_{i,1}$  și  $M_{i,2}$ . Intersecțiile  $(x_{i,1}, y_i), (x_{i,2}, y_i)$  se calculează în planul  $z_s = 0$ , prin intersecția dreptei  $y = y_i$  cu proiecțiile  $A'B'$  și  $B'C'$  ale laturilor AB și BC în planul  $z_s = 0$ . Intersecțiile  $(z_{i,1}, y_i), (z_{i,2}, y_i)$  se calculează în planul  $x_s = 0$ , prin intersecția dreptei  $y = y_i$  cu proiecțiile  $A''B''$  și  $B''C''$  ale laturilor AB și BC în planul  $x_s = 0$ .

Adresele pixelilor se obțin prin rotunjirea (sau trunchierea) coordonatelor punctelor de intersecție  $(x_{i,1}, y_i), (x_{i,2}, y_i)$ ; valorile  $z_{i,1}$  și  $z_{i,2}$  reprezintă adâncimea acestor pixeli, care se află pe conturul poligonului.

Pentru calculul adâncimii pixelilor interiori (a celor ce nu se află pe conturul poligonului), se efectuează o interpolare pe fiecare linie de baleiere.

În planul  $y = y_i$ , relația dintre coordonatele  $x$  și  $z$  ale dreptei  $M_{i,1}M_{i,2}$  este:

$$z = m_{ix}x + n_{ix} \quad (5.7)$$

unde:  $m_{ix} = (x_{i,2} - x_{i,1}) / (z_{i,2} - z_{i,1}), n_{ix} = z_{i,1} - m_{ix} x_{i,1}$

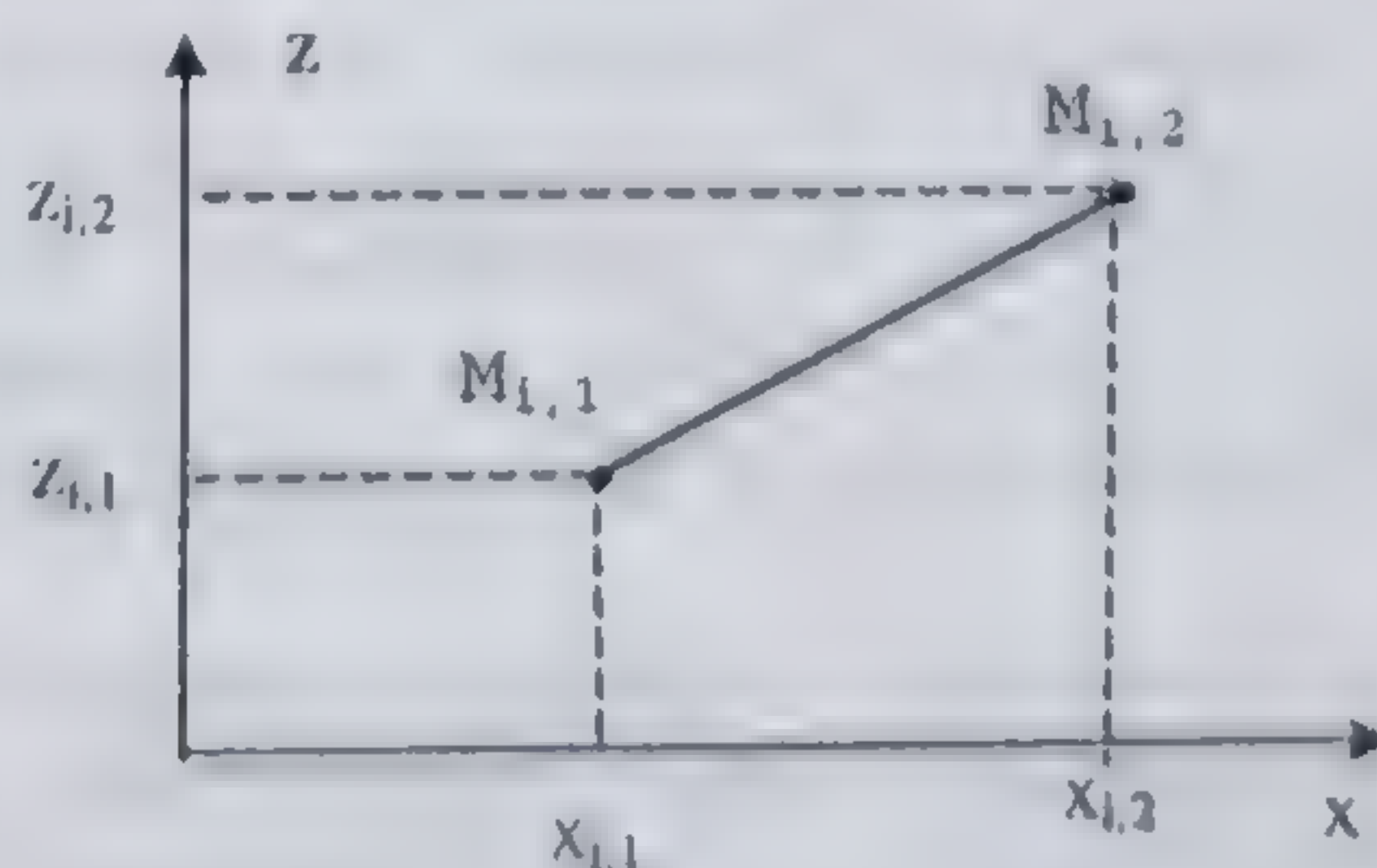


Fig. 5.8 Interpolarea adâncimilor pe linia de baleiere  $y = y_i$ .



Un punct pe această dreaptă are coordonatele  $x_{i,j}$ ,  $z_{i,j} = x_{i,j} m_{ix} + n_{ix}$ . Pe segmentul de dreaptă  $M_{i,1}M_{i,2}$  se calculează puncte succesive prin incrementarea cu 1 a coordonatei  $x$  și calculul prin incrementare cu panta  $m_{ix}$  a coordonatei  $z$ :

$$x_{i,j+1} = x_{i,j} + 1; z_{i,j+1} = z_{i,j} + m_{ix} \quad (5.8)$$

Valoarea  $z_{i,j}$  reprezintă adâncimea pixelului a cărui adresă se obține prin rotunjirea sau trunchierea coordonatelor punctului  $(x_{i,j}, y_i)$ .

Toate aceste operații se execută pentru fiecare față a obiectelor tridimensionale în sistemul de coordonate ecran 3D, și pentru fiecare pixel se actualizează în mod corespunzător culoarea în bufferul de imagine și adâncimea în bufferul de adâncime. Exprimate în pseudocod asemănător limbajului C, operațiile combinate de conversie de baleiere și Z-buffer (funcția scan-line) arată astfel:

```
void scanline(){
    for (toate valorile (x,y)){
        image_buffer[x][y] = culoare_stergere;
        Z_buffer[x][y] = max_z; /* initializ. Z-buffer */
    }
    for (fiecare poligon){
        calculul Y_min, Y_max ale poligonului;
        for (Y_i = Y_min; Y_i <= Y_max; Y_i++){
            calcul x_{i,1}, x_{i,2}
            calcul z_{i,1}, z_{i,2}
            for (x=x_{i,1}; x<x_{i,2}; x++){
                calcul z;
                if (z < Z_buffer[x][Y_i]){
                    Z_buffer[x][Y_i] = z;
                    image_buffer[x][Y_i] = culoare;
                }
            }
        }
    }
}
```

Cel mai important avantaj al algoritmului Z-buffer este simplitatea implementării lui, iar dezavantajul principal îl constituie memoria suplimentară necesară pentru implementarea bufferului de adâncime. Dimensiunea fiecărei locații (fiecare locație corespunzând unui pixel din imagine) este dată de precizia de reprezentare a adâncimii  $z$ . Datorită compresiei pe care o introduce transformarea de la sistemul de referință de observare la sistemele de referință normalizat și ecran 3D (relația 4.10), este posibil ca două valori  $z_v$  distincte să fie transformate în valori  $z_s$  identice. În mod obișnuit, se folosește 20 sau 32 de biți pentru reprezentarea adâncimii unui pixel, ceea ce înseamnă că Z-bufferul poate deveni mai mare decât bufferul de imagine, construit în mod obișnuit cu 24 de biți/pixel.

În realizările mai vechi ale sistemelor grafice, bufferul de adâncime era implementat în memoria principală. În momentul actual, sunt disponibile sisteme grafice care au Z-bufferul realizat ca o memorie dedicată, și aceasta reprezintă cea mai bună soluție.



Un alt avantaj al Z-bufferului îl reprezintă independența de forma de reprezentare a obiectelor. Descrierea dată mai sus se referă la utilizarea Z-bufferului în redarea obiectelor modelate prin suprafețe poligonale, dar aceasta nu este o restricție, deoarece se poate calcula adâncimea  $z$  pentru orice tip de suprafață, iar mecanismul de comparație a adâncimilor este același.

#### 5.3.4 ELIMINAREA SUPRAFETELOR ORIENTATE INVERS

Algoritmii de eliminare a suprafețelor ascunse prezentați anterior se ocupă de compararea suprafețelor provenind de la oricâte obiecte diferite, pentru care trebuie să fie calculate adâncimile și sortate (în spațiul obiect) sau comparate (în spațiul imagine) pentru a decide care suprafață ascunde o altă suprafață.

Există însă și o situație de ascundere între suprafețe care se poate rezolva mult mai simplu, și anume ascunderea reciprocă între fețele unui obiect opac.

În modelarea obiectelor tridimensionale, fețele componente sunt reprezentate orientat în spațiu, astfel încât normalele tuturor fețelor unui obiect să fie îndreptate către aceeași regiune a spațiului (spre interiorul sau spre exteriorul obiectului).

Dacă un obiect opac are a cărei normale la fețe sunt îndreptate spre exterior, este privit dintr-un punct de observare, fețele ale căror normale sunt orientate către punctul de observare ascund (maschează) fețele ale căror normale sunt orientate în direcție inversă.

Orientarea consistentă a fețelor unui obiect permite selecția numai a acelor fețe ale obiectului care sunt orientate către punctul de observare. Această operație de selecție (numită eliminarea fețelor orientate invers - "din spate" - *backface elimination*) se efectuează în sistemul de referință de observare (fig. 5.9).

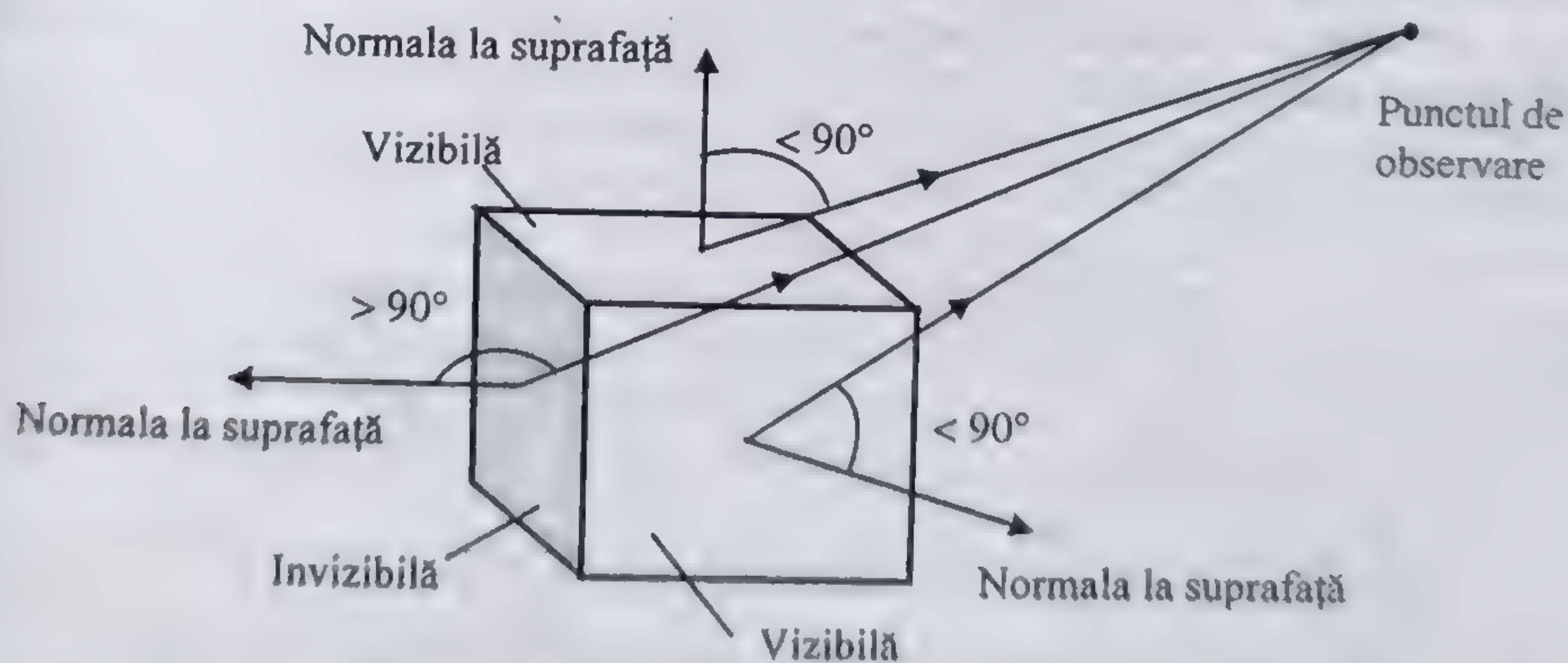


Fig. 5.8 Selecția fețelor în funcție de orientare.

Pentru testul de vizibilitate al fețelor în funcție de orientare se calculează produsul scalar dintre normala la suprafață și un vector dintr-un punct al feței spre punctul de observare. Dacă normalele la fețele obiectului sunt îndreptate către



exterior (sens pozitiv, invers acelor de ceas) și unghiul dintre cei doi vectori este mai mic de  $90^\circ$ , atunci suprafața este vizibilă.

Produsul scalar a doi vectori,  $N$  și  $V$  este:

$$V \cdot N = |V||N|\cos\theta$$

unde  $|V|$  și  $|N|$  sunt modulele celor doi vectori, iar  $\theta$  este unghiul dintre ei. Se poate scrie deci:

$$V \cdot N > 0 \text{ dacă } \theta < 90^\circ$$

$$V \cdot N = 0 \text{ dacă } \theta = 90^\circ$$

$$V \cdot N < 0 \text{ dacă } \theta > 90^\circ$$

De aici rezultă condiția de vizibilitate a unei fețe: pentru ca o față să fie vizibilă, produsul scalar dintre normala acesteia și un vector dintr-un punct al feței spre punctul de observare trebuie să fie pozitiv:

$$N \cdot V > 0 \quad (5.9)$$

Dacă se consideră normala la suprafață orientată către interiorul obiectului (în sens negativ, în direcția acelor de ceas), atunci condiția de vizibilitate a unei fețe este ca produsul scalar dintre normală și un vector dintr-un punct al feței spre punctul de observare să fie negativ.

Eliminarea fețelor orientate invers se execută în sistemul de referință de observare; orice față care este invizibilă este ignorată pentru operațiile ulterioare de redare. Acest test este foarte important deoarece, în medie, jumătate din fețele unui obiect tridimensional sunt orientate invers față de punctul de observare și pot fi eliminate din calcul.

Dacă obiectul are fețe transparente, atunci nu se pot elimina fețele în funcție de orientare și toate fețele obiectului trebuie luate în considerare și prelucrate corespunzător. Pentru o față dată prin succesiunea vârfurilor sale  $V_1, V_2, \dots, V_n$ , normala se calculează ca produs vectorial al vectorilor care reprezintă două laturi succesive (fig. 5.9 (a)).

$$N = (V_2 - V_1) \times (V_3 - V_2)$$

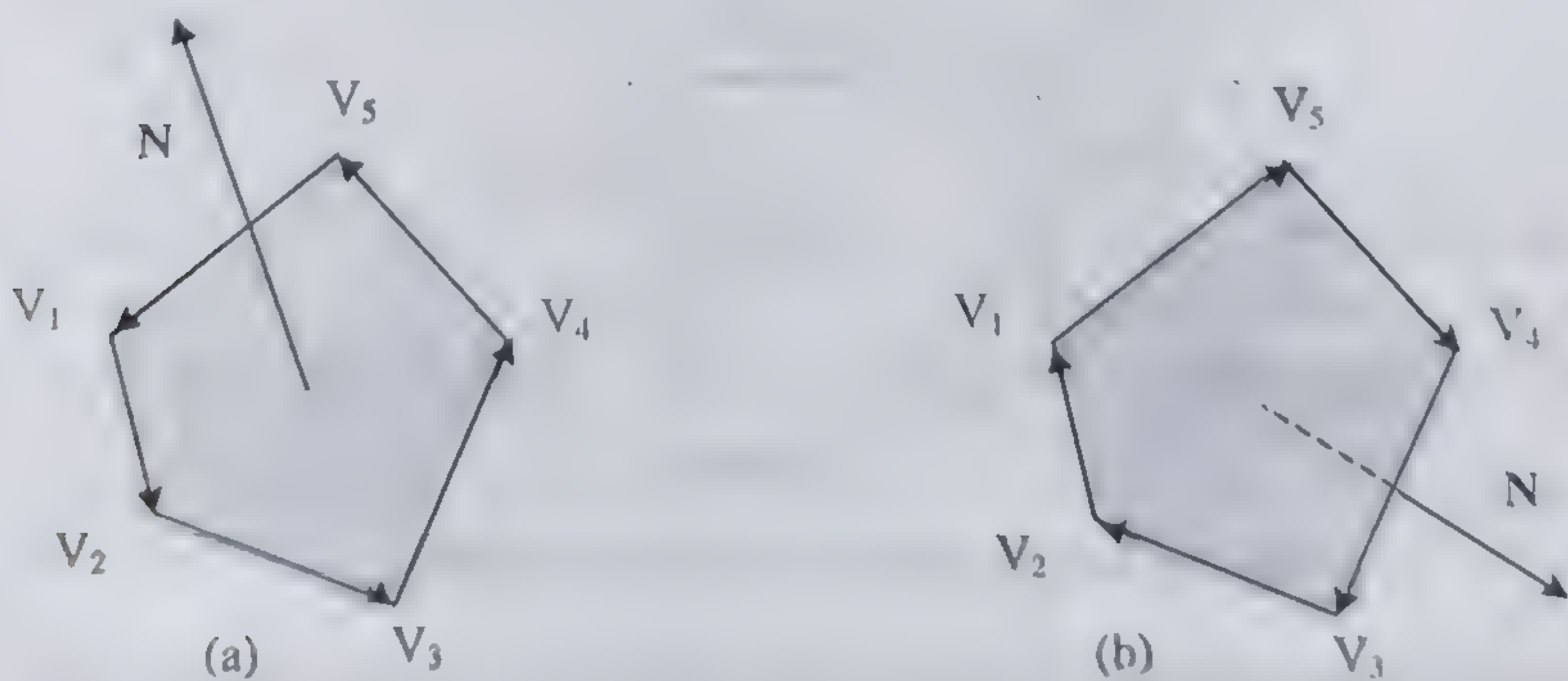


Fig. 5.9 Calculul normalei la o suprafață.



Dacă  $(x_m, y_m, z_m)$  reprezintă coordonatele vârfului  $V_m$ , iar  $i, j, k$  sunt versorii axelor  $x, y, z$ , atunci normala  $N$  are expresia:

$$N = (V_2 - V_1) \times (V_3 - V_2) = \begin{vmatrix} i & j & k \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \end{vmatrix}$$

Se dezvoltă determinantul după prima linie, și rezultă:

$$N = ((y_2 - y_1)(z_3 - z_2) - (y_3 - y_2)(z_2 - z_1))i + ((x_3 - x_2)(z_2 - z_1) - (x_2 - x_1)(z_3 - z_2))j + ((x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1))k \quad (5.10)$$

Dacă ordinea de parcurgere a vârfurilor feței se inversează, deci se consideră fața  $V_n \dots V_2 V_1$ , atunci normala se calculează prin produsul  $N = (V_2 - V_3) \times (V_1 - V_2)$  care are semnul inversat (fig. 5.10 (b)), ceea ce se regăsește în formula de calcul a produsului vectorial.

Rezultă că sensul vectorului normal la o față este dat de ordinea de parcurgere a vârfurilor feței. Dacă în sistemul de referință local (de modelare) fețele unui obiect sunt definite (prin ordinea vârfurilor) astfel încât toate normalele să fie îndreptate spre aceeași regiune a spațiului (spre exteriorul obiectului), această orientare se păstrează după transformarea de instanțiere și observare, și în sistemul de referință de observare obiectele vor avea normalele orientate consistent (toate orientate către exterior). Această condiție este suficientă pentru calculul vizibilității fiecărei fețe prin produs scalar al normalei cu vectorul de direcție de la un punct al feței către punctul de observare.

Eliminarea suprafețelor orientate invers ale obiectelor opace trebuie efectuată în orice sistem de vizualizare, deoarece este un test care se execută eficient și elimină în medie jumătate din suprafețele obiectelor din scenă.

Dacă scena este compusă dintr-un singur obiect convex opac, eliminarea fețelor orientate invers este suficientă pentru a rezolva problema ascunderii fețelor invizibile. Dacă, însă, obiectul este concav, este posibil ca o față orientată direct (către punctul de observare) să fie parțial mascată de o față orientată invers și atunci eliminarea suprafețelor orientate invers nu mai este suficientă, fiind necesar unul din algoritmi de eliminare a suprafețelor ascunse care se bazează pe calculul adâncimii. La fel, dacă în scenă există mai mult de un obiect, pe lângă eliminarea suprafețelor orientate invers este necesară eliminarea suprafețelor ascunse.

### 5.3.5 ASCUNDEREA SUPRAFETELOR COPLANARE

Un caz special de ascundere între suprafețe este cazul suprafețelor coplanare. Suprafețele coplanare sunt necesare în modelarea obiectelor atunci când pe o suprafață dată a unui obiect trebuie să fie suprapusă o altă suprafață. Astfel de situații se întâlnesc în reprezentarea pistelor din aeroporturi sau reprezentarea marcajelor aplicate diferitelor obiecte.



În algoritmi de eliminare a suprafețelor ascunse în spațiul obiect se poate impune condiția ca suprafața aplicată să fie introdusă în lista de display după suprafața suport și în acest fel suprafața aplicată să fie vizibilă.

În algoritmi de eliminare a suprafețelor ascunse în spațiul imagine nu se poate selecta care suprafață este vizibilă dintr-un punct de observare dat, deoarece adâncimile a două suprafețe coplanare sunt egale în toate punctele comune. De aceea sunt necesare tehnici speciale pentru selectarea suprafeței vizibile.

Una din posibilități este de a modifica poziția suprafeței aplicate prin translație cu un vector de translație perpendicular pe suprafața suport și valoarea a modulului astfel aleasă încât să permită discriminarea adâncimii și funcționarea corectă a algoritmului de Z-buffer.

O altă posibilitate este folosirea unui buffer șablon (*stencil buffer*). Un buffer șablon este un buffer de dimensiune egală cu bufferul de imagine ca număr de pixeli reprezentabili. În fiecare locație a bufferului șablon se înscrie o valoare care modifică comportarea algoritmului Z-buffer: un pixel nou calculat înlocuiește pixelul existent în bufferul de imagine nu în funcție de adâncime (memorată în poziția corespunzătoare în Z-buffer), ci în funcție de valoarea memorată în bufferul șablon. În biblioteca OpenGL este prevăzut un buffer șablon care poate fi utilizat pentru redarea suprafețelor coplanare.



## BIBLIOTECA GRAFICĂ OPENGL

Pentru programarea aplicațiilor grafice complexe necesare în crearea și redarea scenelor virtuale, se pot utiliza în momentul de față mai multe biblioteci și interfețe grafice, precum și sisteme de dezvoltare de programe (toolkit-uri), care permit proiectantului să reutilizeze un număr mare de funcții grafice deja implementate și să-și concentreze eforturile asupra aplicației însăși. Dat fiind că majoritatea acestor biblioteci sunt foarte ieftine sau accesibile gratis prin Internet și pot fi folosite într-un număr mare de platforme hardware și software, cunoașterea și utilizarea lor este deosebit de importantă și utilă.

Dintre bibliotecile grafice existente, biblioteca OpenGL, scrisă în limbajul C, este una dintre cele mai utilizate, datorită faptului că implementează un mare număr de funcții grafice de bază pentru crearea aplicațiilor grafice interactive, asigurând o interfață independentă de platforma hardware.

Introdusă în anul 1992, biblioteca OpenGL a devenit în momentul de față cea mai intens folosită interfață grafică pentru o mare varietate de platforme hardware și pentru o mare varietate de aplicații grafice 2D și 3D, de la proiectare asistată de calculator (CAD), la animație, imagistică medicală și realitate virtuală. Datorită suportului oferit prin specificațiile unui consorțiu specializat (OpenGL Architecture Review Board) și numărului mare de implementări existente, în momentul de față OpenGL este în mod real singurul standard grafic multiplatformă. Calitățile de maturitate, stabilitate, portabilitate și fiabilitate se adaugă la cele de compatibilitate între versiunile existente și de ușurință de utilizare datorită unei bune structurări și documentației care se găsește cu ușurință. Au fost publicate numeroase cărți, exemple de cod și alte informații ieftine sau accesibile liber pe rețeaua Internet. Una din adresele Internet care se pot folosi pentru astfel de informații este [www.opengl.org](http://www.opengl.org).

Aplicațiile OpenGL pot rula pe platforme foarte variate, începând cu PC, stații de lucru și până la supercalculatoare, sub cele mai cunoscute sisteme de operare: Linux, Unix, Windows NT, Windows 95, Mac OS. De asemenea se integrează în orice sistem de administrare a ferestrelor (windowing), incluzând Win32, X/Windows, și Presentation Manager. Funcțiile OpenGL sunt apelabile din



limbajele Ada, C, C++ și Java. Ca rezultat, aplicațiile scrise pentru OpenGL pot fi adaptate pentru orice sistem disponibil al utilizatorilor. Din consorțiul de arhitectură OpenGL fac parte reprezentanți de la firmele Compaq, Evans-Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Microsoft și Silicon Graphics. Pentru scrierea aplicațiilor folosind interfața OpenGL nu este necesară obținerea unei licențe de către utilizatori finali.

Implementările bibliotecii OpenGL diferă de la un sistem grafic la altul, dezvoltatorii sistemelor având astfel posibilitatea de optimiza costul și performanțele acestora. Funcțiile OpenGL, care sunt aceleași ca interfață de apel, pot fi executate de hardware specializat, pot fi executate ca rutine soft de procesorul unității centrale CPU, sau pot fi implementate ca o combinație de rutine software și hardware specializat. Această flexibilitate de implementare a permis realizarea unei game variate de acceleratoare grafice OpenGL, de la simple plăci de redare grafică, până la implemetarea completă a pipeline-ului grafic, inclusiv funcțiile de Z-buffer, anti-aliasing, ceață, texturare. Există acceleratoare grafice OpenGL de la plăci cu preț scăzut pentru calculatoare PC, până la sisteme grafice ale stațiilor de lucru puternice și multiprocesoare.

În lucrarea de față sunt prezentate aspectele cele mai importante ale programării aplicațiilor grafice folosind biblioteca OpenGL, cu intenția de a asigura un minim de cunoștințe de pornire a experienței de programare. De asemenea, sunt descrise programele folosite pentru exemplificarea aspectelor teoretice de modelare și generare a imaginilor (prezentate în capitolele 2, 3, 4 și 5). Prezentarea completă a tuturor funcțiilor bibliotecii OpenGL nu este nici posibilă, nici necesară. Numai ghidul de programare OpenGL (*OpenGL Programming Guide* [Woo97]), conține peste 600 de pagini, la care se adaugă manualele de referință și multe altele. Dar, odată înțelese principiile de organizare a interfeței, restul detaliilor de implementare se pot parcurge cu ușurință din orice material disponibil sau help on-line.

Folosind biblioteca OpenGL, un programator în domeniul graficii se poate concentra asupra aplicației dorite și nu mai trebuie să fie preocupat de detaliile de implementare a diferitelor funcții "standard". Programatorul nu este obligat să scrie algoritmul de generare a liniilor sau suprafețelor, nu trebuie să calculeze decuparea obiectelor la volumul de vizualizare etc., toate acestea fiind deja implementate în funcțiile bibliotecii și, cel mai probabil, mult mai eficient și adaptat platformei hardware decât le-ar putea realiza el însuși.

În redarea obiectelor tridimensionale, biblioteca OpenGL folosește un număr redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor și scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicație, pe baza acestor primitive. Dat fiind că OpenGL prevede un set puternic, dar de nivel scăzut, de comenzi de redare a obiectelor, mai sunt folosite și alte biblioteci de nivel mai înalt care utilizează aceste comenzi și preiau o parte din sarcinile de programare grafică. Astfel de biblioteci sunt:

- Biblioteca de funcții utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafețelor curbe și alte funcții grafice.



- Pentru fiecare sistem Windows există o extensie a bibliotecii OpenGL care asigură interfața cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc sistemul X Window, extensia GLX, pentru sisteme IBM OS/2, extensia PGL.
- Biblioteca de dezvoltare GLUT (OpenGL Utility Toolkit) este un sistem de dezvoltare independent de sistemul Windows, care ascunde dificultățile interfețelor de aplicații Windows, punând la dispoziție funcții pentru crearea și inițializarea ferestrelor, funcții pentru prelucrarea evenimentelor de intrare și pentru execuția programelor grafice bazate pe biblioteca OpenGL.

În orice aplicație OpenGL trebuie să fie incluse fișierele header `gl.h` și `glu.h` sau `glut.h` (dacă este utilizată biblioteca GLUT). Toate funcțiile bibliotecii OpenGL încep cu prefixul `gl`, funcțiile GLU încep cu prefixul `glu`, iar funcțiile GLUT încep cu prefixul `glut`.

## 6.1 DEZVOLTAREA PROGRAMELOR GRAFICE FOLOSIND UTILITARUL GLUT

Biblioteca grafică OpenGL conține funcții de redare a primitivelor geometrice independente de sistemul de operare, de orice sistem Windows sau de platforma hardware. Ea nu conține nici o funcție pentru a crea sau administra ferestre de afișare pe display (windows) și nici funcții de citire a evenimentelor de intrare (mouse sau tastatură).

Pentru crearea și administrarea ferestrelor de afișare și a evenimentelor de intrare se pot aborda mai multe soluții: utilizarea directă a interfeței Windows (Win32 API), folosirea compilatoarelor sub Windows, care conțin funcții de acces la ferestre și evenimente sau folosirea altor instrumente (utilitare) care înglobează interfața OpenGL în sistemul de operare respectiv.

Un astfel de utilitar este GLUT, care se compilează și instalează pentru fiecare tip de sistem Windows. Header-ul `glut.h` trebuie să fie inclus în fișierele aplicației, iar biblioteca `glut.lib` trebuie legată (linkată) cu programul de aplicație. În lucrarea de față s-a folosit versiunea `glut3.6`, care poate fi preluată din Internet ([www.sgi.com/pub/opengl/GLUT](http://www.sgi.com/pub/opengl/GLUT)).

Sub GLUT, orice aplicație se structurează folosind mai multe funcții callback. O funcție callback este o funcție care aparține programului aplicației și este apelată de un alt program, în acest caz sistemul de operare, la apariția anumitor evenimente. În GLUT sunt predefinite câteva tipuri de funcții callback pentru inițializarea programului, redimensionarea ferestrei de afișare, desenarea ferestrei și controlul dispozitivelor de intrare (tastatură și mouse). Aceste funcții sunt scrise în aplicație și pointerii lor sunt transmiși la înregistrare sistemului Windows, care le apelează (prin pointerul primit) în momentele necesare ale execuției.



### 6.1.1 FUNCȚII DE CONTROL AL FERESTREI DE AFIȘARE

Sunt disponibile cinci funcții pentru controlul ferestrei de afișare a programului. Aici trebuie să fie precizate diferitele semnificații ale termenului suprautilizat "fereastră". În general, în programare se folosește termenul fereastră (*window*) pentru orice zonă de afișare controlată pe display; ferestrele sunt gestionate de sistemul de operare (în cazul sistemului Microsoft Windows (95, NT) sau de subsistemul grafic X Window, care se execută sub sistemele de operare de tip Unix.

În aplicațiile de grafică tridimensională termenul de fereastră de vizualizare (*view plane window*) se referă la regiunea rectangulară care reprezintă intersecția dintre planul de vizualizare și volumul de vizualizare și în care sunt proiectate toate obiectele vizibile ale scenei. Fereastra de vizualizare se afișează într-o poartă de afișare (*viewport*), care se definește prin transformarea fereastră-poartă. Poarta de afișare se amplasează într-o fereastră de afișare (*window*) și legătura dintre acestea este realizată diferit, în funcție de modul de programare folosit (interfață Windows sau toolkit).

În programele dezvoltate sub GLUT, pentru corelarea dintre poarta de afișare și fereastra de afișare se folosesc funcțiile `glutInit()` și `glutInitDisplayMode()`. Funcția:

```
void glutInit(int* argc, char** argv);
```

inițializează biblioteca GLUT folosind argumentele din linia de comandă; ea trebuie să fie apelată înaintea oricăror alte funcții GLUT sau OpenGL. Funcția:

```
void glutInitDisplayMode(unsigned int mode);
```

specifică caracteristicile de afișare a culorilor și a bufferului de adâncime și numărul de buffere de imagine. Parametrul `mode` se obține prin SAU logic între valorile fiecărei opțiuni. De exemplu:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
```

inițializează afișarea culorilor în modul RGB, cu două buffere de imagine și buffer de adâncime. Alte valori ale parametrului `mode` sunt: `GLUT_SINGLE` (un singur buffer de imagine), `GLUT_RGBA` (modelul RGBA al culorii), `GLUT_STENCIL` (validare buffer șablon), `GLUT_ACCUM` (validare buffer de acumulare).

Funcția:

```
void glutInitWindowPosition(int x, int y);
```

specifică poziția pe ecran a colțului stânga sus al ferestrei de afișare. Pentru definirea dimensiunii inițiale a ferestrei de afișare se apelează funcția:

```
void glutInitWindowSize(int width, int height);
```

Crearea ferestrei în care se afișează contextul de redare (poarta) OpenGL are loc la apelul funcției:

```
int glutCreateWindow(char* string);
```



### 6.1.2 FUNCȚII CALLBACK

Funcțiile callback se definesc în program și se înregistrează în sistem prin intermediul unor funcții GLUT. Ele sunt apelate de sistemul de operare atunci când este necesar, în funcție de evenimentele apărute. Funcția:

```
glutDisplayFunc(void(*Display)(void));
```

înregistrează funcția callback `Display()` care calculează și afișează imaginea. Argumentul funcției este un pointer la o funcție fără argumente care nu returnează nici o valoare. Funcția `Display` (a aplicației) este apelată oridecâte ori este necesară desenaarea ferestrei: la inițializare, la modificarea dimensiunilor ferestrei, sau la apelul explicit al funcției `glutPostRedisplay()`. Funcția:

```
glutReshapeFunc(void(*Reshape)(int w, int h));
```

înregistrează funcția callback `Reshape()` care este apelată oridecâte ori se modifică dimensiunea ferestrei de afișare. Argumentul este un pointer la funcția cu numele `Reshape` cu două argumente de tip întreg și care nu returnează nici o valoare. În această funcție, programul de aplicație trebuie să refacă transformarea fereastră-poartă, dat fiind că fereastra de afișare și-a modificat dimensiunile.

Funcția:

```
glutKeyboardFunc(void(*Keyboard)(unsigned int key,  
int x, int y));
```

înregistrează funcția callback `Keyboard()` care este apelată atunci când se acționează o tastă. Parametrul `key` este codul tastei, iar `x` și `y` sunt coordonatele (relativ la fereastra de afișare) a mouse-ului în momentul acționării tastei.

Funcția:

```
glutMouseFunc(void(*MouseFunc)(unsigned int button,  
int state, int x, int y));
```

înregistrează funcția callback `MouseFunc` care este apelată atunci când este apăsat sau eliberat un buton al mouse-ului. Parametrul `button` este codul butonului (poate avea una din constantele `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` sau `GLUT_RIGHT_BUTTON`). Parametrul `state` indică apăsarea (`GLUT_DOWN`) sau eliberarea (`GLUT_UP`) al unui buton al mouse-ului. Parametrii `x` și `y` sunt coordonatele relativ la fereastra de afișare a mouse-ului în momentul evenimentului.

Execuția unui program folosind toolkit-ul GLUT se lansează prin apelul funcției `glutMainLoop()` după ce au fost efectuate toate inițializările și înregistrările funcțiilor callback. Această buclă de execuție poate fi oprită prin închiderea ferestrei aplicației. În cursul execuției sunt apelate funcțiile callback în momentele apariției diferitelor evenimente. În cursul execuției se mai poate executa un proces atunci când nu apare nici un eveniment (atunci când coada de evenimente este vidă). Acest proces, folosit în animație, execută funcția callback `IdleFunc()` înregistrată prin apelul funcției:

```
glutIdleFunc(void(*IdleFunc)(void));
```



### 6.1.3 GENERAREA OBIECTELOR TRIDIMENSIONALE

Multe programe folosesc modele simple de obiecte tridimensionale pentru a ilustra diferite aspecte ale prelucrărilor grafice. GLUT conține câteva funcții care permit redarea unor astfel de obiecte tridimensionale în modul wireframe sau cu suprafețe pline (*filled*). Fiecare obiect este reprezentat într-un sistem de referință local, iar dimensiunea lui poate fi transmisă ca argument al funcției respective. Poziționarea și orientarea obiectelor în scenă se face în programul de aplicație. Exemple de astfel de funcții sunt:

```
void glutWireCube(GLdouble size);
void glutSolidCube(GLdouble size);
void glutWireSphere(GLdouble radius, GLint slices,
                    GLint stacks);
void glutSolidSphere(GLdouble radius, GLint slices,
                    GLint stacks);
```

Programele GLUT au un mod specific de organizare, care provine din felul în care sunt definite și apelate funcții callback. Acest mod va fi prezentat la primul exemplu de program OpenGL-GLUT și va fi reluat apoi și în alte exemple.

## 6.2 CARACTERISTICILE BIBLIOTECII OPENGL

În fișierul header ale bibliotecii OpenGL (gl.h) sunt definite mai multe constante simbolice, care reprezintă diferite stări, variabile sau valori de selecție a opțiunilor OpenGL. Aceste constante sunt toate scrise cu majuscule și sunt precedate de prefixul GL\_. De exemplu, constantele simbolice care definesc valorile TRUE și FALSE și cele care selectează tipul unei primitive grafice sunt:

```
/* Boolean */
#define GL_TRUE 1
#define GL_FALSE 0

/* BeginMode */
#define GL_POINTS 0x0000
#define GL_LINES 0x0001
#define GL_LINE_LOOP 0x0002
#define GL_LINE_STRIP 0x0003
#define GL_TRIANGLES 0x0004
#define GL_TRIANGLE_STRIP 0x0005
#define GL_QUADS 0x0007
#define GL_POLYGON 0x0009
```

De asemenea, biblioteca OpenGL definește propriile tipuri de date, cele mai multe corespunzând tipurilor de date fundamentale ale limbajului C. De exemplu, în gl.h sunt definite următoarele tipuri:



```

typedef unsigned int GLenum;
typedef unsigned char GLboolean;
typedef signed char GLbyte;
typedef unsigned char GLubyte;
typedef short GLshort;
typedef int GLint;
typedef int GLsizei;
typedef unsigned int GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
typedef double GLclampd;
typedef void GLvoid;

```

Pentru a înțelege funcționarea comenzilor OpenGL, se descriu în continuare cele mai importante dintre caracteristicile OpenGL.

### 6.2.1 POARTA DE AFIȘARE OPENGL

Poarta de afișare OpenGL mai este numită *context de redare (rendering context)* și este asociată unei ferestre de afișare din sistemul Windows. Dacă se programează folosind biblioteca GLUT, corelarea dintre fereastra de afișare și poarta OpenGL este asigurată de funcții ale acestei biblioteci. Dacă nu se folosește biblioteca GLUT, atunci funcțiile bibliotecilor de extensie XGL, WGL sau PGL permit asocierea contextului de redare OpenGL cu o fereastră de afișare și accesul la aceasta. Funcția OpenGL care definește transformarea fereastră-poartă este:

```

void glViewport(GLint x, GLint y,
                GLsizei width, GLsizei height);

```

unde  $x$  și  $y$  specifică poziția colțului stânga-jos al dreptunghiului porții în fereastra de afișare (window) și au valorile implicite 0, 0. Parametrii *width* și *height* specifică lățimea, respectiv înălțimea, porții de afișare, dată ca număr de pixeli.

În OpenGL un pixel este reprezentat printr-un descriptor care definește mai mulți parametri:

- numărul de biți/pixel pentru memorarea culorii
- numărul de biți/pixel pentru memorarea adâncimii
- numărul de buffere de imagine.

### 6.2.2 BUFFERUL DE CADRU

Bufferul de cadru (*frame buffer*) conține toate datele care definesc o imagine și constă din mai multe secțiuni logice: bufferul de imagine (sau bufferul de culoare), bufferul de adâncime (*Z-buffer*), bufferul șablon (*stencil*), bufferul de acumulare (*accumulation*).

**Bufferul de imagine** (*image buffer, color buffer*) în OpenGL poate conține una sau mai multe secțiuni, în fiecare fiind memorată culoarea pixelilor din poarta de afișare. Redarea imaginilor folosind un singur buffer de imagine este folosită



pentru imagini statice, cel mai frecvent în proiectarea grafică (CAD). În generarea interactivă a imaginilor dinamice, un singur buffer de imagine produce efecte nedorite, care diminuează mult calitatea imaginii generate.

Orice cadru de imagine începe cu ștergerea (de fapt, umplerea cu o culoare de fond) a bufferului de imagine. După aceasta sunt generați pe rând pixelii care aparțin tuturor elementelor imaginii (linii, puncte, suprafețe) și intensitățile de culoare ale acestora sunt înscrise în bufferul de imagine. Pentru trecerea la cadrul următor, trebuie din nou șters bufferul de imagine și reluată generarea elementelor componente, pentru noua imagine. Chiar dacă ar fi posibilă generarea și înscrisura în buffer a elementelor imaginii cu o viteză foarte mare (ceea ce este greu de realizat), tot ar exista un interval de timp în care bufferul este șters și acest lucru este perceput ca o pâlpâire a imaginii. În grafica interactivă timpul necesar pentru înscrisura datelor în buffer este (în cazul cel mai fericit) foarte apropiat de intervalul de schimbare a unui cadru a imaginii (update rate) și, dacă acest proces are loc simultan cu extragerea datelor din buffer și afișarea lor pe display, atunci ecranul va prezenta un timp foarte scurt imaginea completă a fiecărui cadru, iar cea mai mare parte din timp ecranul va fi șters sau va conține imagini parțiale ale cadrului. Tehnica universal folosită pentru redarea imaginilor dinamice (care se schimbă de la un cadru la altul) este tehnica *dublului buffer de imagine*.

În această tehnică există două buffere de imagine: bufferul din față (*front*), din care este afișată imaginea pe ecran și bufferul din spate (*back*), în care se înscriu elementele de imagine generate. Când imaginea unui cadru a fost complet generată, ea poate fi afișată pe ecran printr-o simplă operație de comutare între buffere: bufferul spate devine buffer față, și din el urmează să fie afișată imagine cadrului curent, iar bufferul față devine buffer spate și în el urmează să fie generată imaginea noului cadru. Comutarea între bufferele de imagine se poate sincroniza cu cursa de revenire pe verticală a monitorului și atunci imaginile sunt prezentate continuu, fără să se observe imagini fragmentate sau pâlpâiri.

În OpenGL comutarea bufferelor este efectuată de funcția `SwapBuffers()`. Această funcție trebuie apelată la terminarea generării imaginii tuturor obiectelor vizibile în fiecare cadru.

Modul în care se execută comutarea bufferelor depinde de platforma hardware. În lipsa unui accelerator grafic, toate operațiile OpenGL sunt executate software, bufferul spate este o zonă din memorie principală, iar bufferul față este memoria video din adaptorul grafic. La comutarea bufferelor care se produce la apelul funcției `SwapBuffers()` are loc copierea bufferului back (care este implementat ca un bitmap independent de dispozitiv - DIB - *device independent bitmap*), în bufferul front, care este memoria de ecran a sistemului grafic.

Biblioteca OpenGL oferă posibilitatea creerii imaginilor cu simplu sau dublu buffer, monoscopice și stereoscopice.

Funcția `void glDrawBuffer(GLenum mode)` stabilește bufferul în care se desenează primitivele geometrice. Parametrul `mode` poate lua una din valorile: `GL_LEFT`, `GL_RIGHT`, `GL_FRONT`, `GL_BACK`, `GL_FRONT_LEFT`, `GL_FRONT_RIGHT`, `GL_BACK_LEFT`, `GL_BACK_RIGHT`, `GL_NONE`.



Valoarea implicită este `GL_FRONT` pentru imagini cu un singur buffer și `GL_BACK` pentru imagini cu dublu buffer. Celelalte opțiuni se referă la imagini stereoscopice.

Funcția `glReadBuffer(GLenum mode)` stabilește bufferul curent din care se citește pixelii sursă în operații de combinare sau acumulare. Parametrul `mode` poate lua una din valorile descrise anterior. Valoarea implicită este `GL_FRONT` pentru imagini monoscopice cu un singur buffer de culoare sau `GL_BACK` pentru imagini monoscopice cu două buffere de culoare.

Pentru imaginile stereoscopice, biblioteca OpenGL pune la dispoziție bufferele necesare pentru definirea celor două imagini diferite (stânga, dreapta), pentru simplu și dublu buffer. Crearea propriu-zisă a imaginilor stereoscopice este rezolvată la nivelul programului de aplicație și necesită dispozitive de afișare adecvate.

**Bufferul de adâncime** (*depth buffer*) memorează adâncimea fiecărui pixel și, prin aceasta, permite eliminarea suprafețelor ascunse. Bufferul de adâncime conține același număr de locații ca și un buffer de imagine, fiecare locație corespunzând unui pixel, de o anumită adresă. Valoarea memorată în locația corespunzătoare unui pixel este distanța față de punctul de observare (adâncimea pixelului). La generarea unui nou pixel cu aceeași adresă, se compară adâncimea noului pixel cu adâncimea memorată în bufferul de adâncime, și noul pixel înlocuiește vechiul pixel (îl "ascunde") dacă este mai apropiat de punctul de observare. Bufferul de adâncime se mai numește și Z-buffer, de la coordonata *z*, care reprezintă adâncimea în sistemul de referință ecran 3D.

**Bufferul șablon** (*stencil buffer*) este un buffer auxiliar utilizat pentru restricționarea desenării în anumite zone ale porții de afișare. Se pot obține astfel imagini încadrate în anumite șabloane, ca de exemplu, instrumente de afișare folosite în simulatoare de antrenament. Bufferul șablon poate fi folosit și pentru redarea suprafețelor coplanare.

**Bufferul de acumulare** (*accumulation buffer*) este folosit pentru crearea imaginilor anti-aliasing prin acumularea intensităților de culoare a pixelilor rezultați prin eșantionarea imaginii.

### 6.2.3 OPERAȚIILE DE BAZĂ OPENGL

OpenGL desenează *primitive geometrice* (puncte, linii și poligoane) în diferite moduri selectabile. Primitivele sunt definite printr-un grup de unul sau mai multe vârfuri (*vertices*). Un vârf definește un punct, capătul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date:

- coordonate
- culoare
- normală
- coordonate de textură.



Aceste date sunt prelucrate independent, în ordine și în același mod pentru toate primitivele geometrice. Singura deosebire care apare este aceea că, dacă o linie sau o suprafață este decupată, atunci grupul de vârfuri care descriu inițial primitiva respectivă este înlocuit cu un alt grup, în care pot apare vârfuri noi rezultate din intersecția laturilor cu planele volumului de decupare (volumul canonic) sau unele vârfuri din cele inițiale pot să dispară.

Comenzile sunt prelucrate în ordinea în care au fost trimise și orice comandă trebuie executată complet înainte ca o nouă comandă să fie executată. OpenGL este o interfață procedurală, secvența de comenzi (apeluri de funcții) descriu ce operații să fie executate și nu ce rezultat s-ar dori obținut. Se pot specifica valori ale matricelor de transformare, tipul proiecției, coeficienți de reflexie, etc.

Modelul de execuție OpenGL este modelul *client-server*. O aplicație (*client*) generează comenzi care sunt interpretate și executate de către OpenGL (*server*). Serverul OpenGL poate să fie executat pe un alt calculator decât aplicația, dar atunci trebuie definit un protocol de comunicație client-server, dat fiind că nu există comenzi OpenGL pentru astfel de operații.

Efectul comenzilor OpenGL asupra bufferului de imagine este controlat de sistemul Windows care alocă zona de afișare a porții (viewport) OpenGL pe ecran.

Fig. 6.1 prezintă o diagramă bloc simplificată a modului în care OpenGL prelucrează datele. S-au ignorat pentru început aspectele legate de texturare și anti-aliasing. Secvența de operații OpenGL (numită *pipeline grafic*) se execută asupra vârfurilor și asupra datelor asociate acestora: coordonate, culoare, normală (și coordonate de textură, care nu sunt reprezentate în figură).

Biblioteca OpenGL primește o succesiune de primitive geometrice pe care le desenează, adică le convertește în mulțimea de pixeli corespunzătoare, înscriind valorile culorilor acestora într-un buffer de imagine. În continuare sunt detaliate fiecare dintre operațiile grafice prezentate în figură.

Modul în care este executată secvența de operații pentru redarea primitivelor grafice depinde de starea bibliotecii OpenGL, stare care este definită prin mai multe variabile de stare ale acesteia (parametri). Numărul de variabile de stare ale bibliotecii este destul de mare, descrierea lor poate fi găsită în manualul de referință (*OpenGL Reference Manual*), iar pe parcursul expunerii vor fi prezentate numai cele mai importante dintre acestea.

La inițializare, fiecare variabilă de stare este setată la o valoare implicită. O stare o dată setată își menține valoarea neschimbată până la o nouă setare. Variabilele de stare au denumiri date sub formă de constante simbolice care pot fi folosite pentru aflarea valorilor acestora. Câteva exemple de stări definite prin constante simbolice în fișierul *gl.h* sunt:

```
#define GL_CURRENT_COLOR          0x0B00
#define GL_CURRENT_NORMAL         0x0B02
#define GL_POLYGON_MODE           0x0B40
#define GL_FOG_COLOR              0x0B66
#define GL_MODELVIEW_MATRIX       0x0BA6
#define GL_PROJECTION_MATRIX      0x0BA7
```



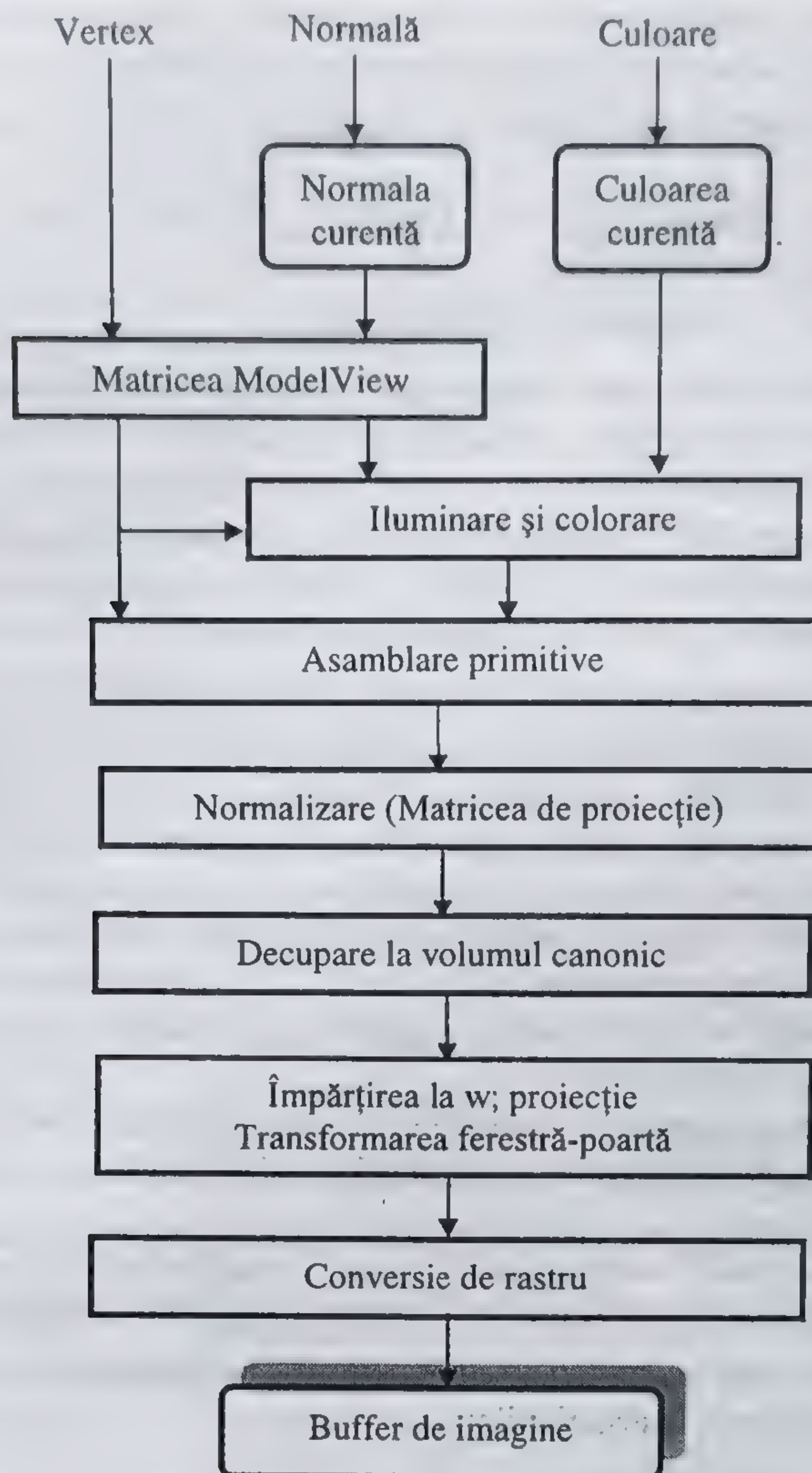


Fig. 6.1 Operațiile grafice OpenGL

Variabilele de stare OpenGL sunt de două categorii: variabile de tip binar și variabile definite prin diferite structuri de date.

Variabile de tip binar pot avea una din două stări: starea activă (*enabled*) sau starea inactivă (*disabled*). Setarea la starea activă se realizează prin apelul funcției:

```
void glEnable(GLenum param);
```

unde *param* este numele simbolic al parametrului (variabilei de stare).

Setarea la starea inactivă se realizează prin apelul funcției:

```
void glDisable(GLenum param);
```



De exemplu, apelul funcției `glEnable(GL_DEPTH_TEST)` activează testul de adâncime și actualizarea corespunzătoare a bufferului de adâncime (*depth buffer*), iar apelul funcției `glDisable(GL_DEPTH_TEST)` dezactivează testul de adâncime.

În orice loc într-un program OpenGL, se poate afla valoarea unui parametru binar prin apelul funcției:

```
GLboolean glIsEnabled(GLenum param);
```

care returnează `GL_TRUE` dacă parametrul `param` este în starea activă și `GL_FALSE` dacă parametrul `param` este în starea inactivă.

Valoarea unei variabile de stare care nu este de tip binar se setează prin apelul unei funcții specifice variabilei respective, care are ca argumente valorile necesare pentru actualizare. De exemplu, variabila de stare culoare curentă, denumită `GL_CURRENT_COLOR`, se setează prin funcția:

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
```

la o valoare dată prin trei componente: roșu (*red*), verde (*green*), albastru (*blue*).

Fiind o bibliotecă dezvoltată în limbajul C, fără posibilitatea de supraîncărcare a funcțiilor, selecția unei funcții apelate cu diferite tipuri de argumente de apel este realizată prin modificarea (printr-un sufix) a numelui funcției. De exemplu, funcția de setare a culorii curente are mai multe variante, după tipul și numărul argumentelor: `glColor3f()`, `glColor4d()`, etc. În continuare, în această lucrare se notează generic cu # sufixul dintr-o familie de funcții (de exemplu, `glColor#()`).

Valoarea unui parametru (variabilă de stare) OpenGL care nu este de tip binar se poate obține în orice loc în program prin apelul unei funcții `glGet#()`:

```
void glGetBooleanv(GLenum pname, GLboolean* params);
void glGetDoublev(GLenum pname, GLdouble* params);
void glGetIntegerv(GLenum pname, GLint* params);
```

unde `params` este adresa unde se depun valorile componente ale parametrului `pname`. De exemplu, prin instrucțiunile:

```
GLfloat color[4];
glGetFloatv(GL_CURRENT_COLOR, color);
```

se obțin cele patru componente ale culorii curente (*red*, *green*, *blue*, *alpha*) în vectorul `color`.

## 6.2.4 PRIMITIVE GEOMETRICE

Funcțiile OpenGL execută secvența de operații grafice asupra fiecărei primitive geometrice, definită printr-o mulțime de vârfuri și tipul acesteia. Coordonatele unui vârf sunt transmise către OpenGL prin apelul unei funcții `glVertex#()`. Aceasta are mai multe variante, după numărul și tipul argumentelor. Iată, de exemplu, numai câteva din prototipurile funcțiilor `glVertex#()`:



```

void glVertex2d(GLdouble x, GLdouble y);
void glVertex2i(GLint x, GLint y);
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
void glVertex4d(GLdouble x, GLdouble y,
                GLdouble z, GLdouble w);

```

Vârfurile pot fi specificate în plan, în spațiu sau în coordonate omogene, folosind apelul funcției corespunzătoare.

O primitivă geometrică se definește printr-o mulțime de vârfuri (care dau descrierea geometrică a primitivei) și printr-unul din tipurile prestabilite, care indică topologia, adică modul în care sunt conectate vârfurile între ele. Mulțimea de vârfuri este delimitată între funcțiile `glBegin()` și `glEnd()`. Aceeași mulțime de vârfuri ( $v_0, v_1, v_2, \dots, v_{n-1}$ ) poate fi tratată ca puncte izolate, linii, poligon, etc, în funcție de tipul primitivei, care este transmis ca argument al funcției `glBegin(GLenum mode)`.

Valorile argumentului `mode` și, deci, tipurile de primitive acceptate de OpenGL sunt prezentate în tabelul de mai jos. În fig. 6.2 sunt ilustrate aceste primitive geometrice.

Tabelul 6.1

### Tipurile de primitive geometrice

Argument	Primitivă geometrică
<code>GL_POINTS</code>	Desenează $n$ puncte
<code>GL_LINES</code>	Desenează segmentele de dreaptă izolate $(v_0, v_1), (v_2, v_3), \dots$ ș.a.m.d. Dacă $n$ este impar, ultimul vârf este ignorat.
<code>GL_LINE_STRIP</code>	Desenează linia poligonală formată din segmentele $(v_0, v_1), (v_1, v_2), \dots, (v_{n-2}, v_{n-1})$ .
<code>GL_LINE_LOOP</code>	Desenează linia poligonală închisă formată din segmentele $(v_0, v_1), (v_1, v_2), \dots, (v_{n-2}, v_{n-1}), (v_{n-1}, v_0)$ .
<code>GL_TRIANGLES</code>	Desenează o serie de triunghiuri folosind vârfurile $(v_0, v_1, v_2), (v_3, v_4, v_5), \dots$ ș.a.m.d. Dacă $n$ nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
<code>GL_TRIANGLE_STRIP</code>	Desenează o serie de triunghiuri folosind vârfurile $(v_0, v_1, v_2), (v_2, v_1, v_3), \dots$ ș.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeași orientare.
<code>GL_TRIANGLE_FAN</code>	Desenează triunghiurile $(v_0, v_1, v_2), (v_0, v_2, v_3), \dots$ ș.a.m.d.
<code>GL_QUADS</code>	Desenează o serie de patrulatere $(v_0, v_1, v_2, v_3), (v_4, v_5, v_6, v_7), \dots$ ș.a.m.d. Dacă $n$ nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
<code>GL_QUADS_STRIP</code>	Desenează o serie de patrulatere $(v_0, v_1, v_3, v_2), (v_3, v_2, v_5, v_4), \dots$ ș.a.m.d. Dacă $n < 4$ , nu se desenază nimic. Dacă $n$ este impar, ultimul vârf este ignorat.
<code>GL_POLYGON</code>	Desenează un poligon cu $n$ vârfuri, $(v_0, v_1, \dots, v_{n-1})$ . Dacă poligonul nu este convex, rezultatul este impredictibil.



Acest tabel, împreună cu fig. 6.2 sunt suficiente pentru înțelegerea modului în care OpenGL interpretează primitivele geometrice.

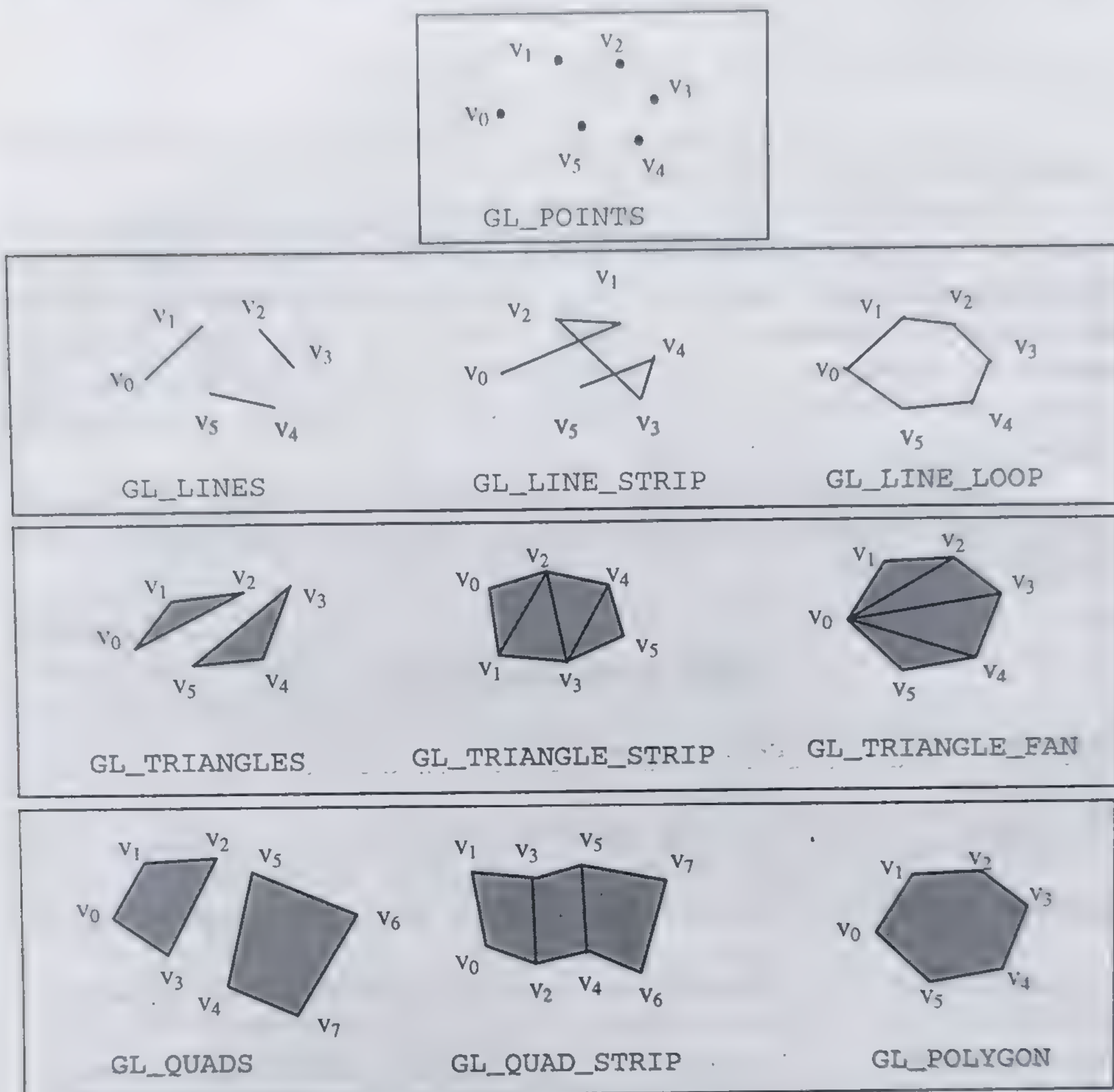


Fig. 6.2 Primitivele geometrice OpenGL.

Primitivele de tip suprafață (triunghiuri, patrulatere, poligoane) pot fi desenate în modul "cadru de sârmă" (*wireframe*), sau în modul "plin" (*fill*), prin setarea variabilei de stare `GL_POLYGON_MODE` folosind funcția

```
void glPolygonMode(GLenum face, GLenum mode);
```

unde argumentul `mode` poate lua una din valorile:

- `GL_POINT` : se desenează numai vârfurile primitivei, ca puncte în spațiu, indiferent de tipul acesteia.
- `GL_LINE`: muchiile poligoanelor se desenează ca segmente de dreaptă.
- `GL_FILL`: se desenează poligonul plin.



Argumentul `face` se referă la tipul primitivei geometrice (din punct de vedere al orientării), căreia i se aplică modul de redare `mode`. Din punct de vedere al orientării, OpenGL admite primitive orientate direct și primitive orientate invers. Argumentul `face` poate lua una din valorile: `GL_FRONT`, `GL_BACK` sau `GL_FRONT_AND_BACK`, pentru a se specifica primitive orientate direct, primitive orientate invers și, respectiv, ambele tipuri de primitive.

În mod implicit, sunt considerate orientate direct suprafețele ale căror vârfuri sunt parcurse în ordinea inversă acelor de ceas. Această setare se poate modifica prin funcția `glFrontFace(GLenum mode)` unde `mode` poate lua valoarea `GL_CCW` pentru orientare în sens invers acelor de ceas (*counterclockwise*) sau `GL_CW` pentru orientare în sensul acelor de ceas (*clockwise*).

### 6.2.5 REPREZENTAREA CULORILOR ÎN OPENGL

În biblioteca OpenGL sunt definite două modele de culori: modelul de culori RGBA și modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B și transparența A pentru fiecare primitivă geometrică sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentată printr-un index într-o tabelă de culori (*color map*), care are memorate pentru fiecare intrare (index) componentele corespunzătoare R,G,B,A ale culorii. În modul de culori indexate numărul de culori afișabile simultan este limitat de dimensiunea tabelii culorilor și, în plus, nu se pot efectua unele dintre prelucrările grafice importante (cum sunt umbrirea, anti-aliasing, ceața). Modelul de culori indexate este folosit în principal în aplicații de proiectare grafică (CAD), în care este necesar un număr mic de culori și nu se folosesc umbrirea, ceața, etc. În aplicațiile de realitate virtuală nu se poate folosi modelul de culori indexate și de aceea în continuare nu vor mai fi prezentate comenzile sau opțiunile care se referă la acest model și toate descrierile consideră numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitivă geometrică depinde de mai multe condiții, putând fi o culoare constantă a primitivei, o culoare calculată prin interpolare între culorile vârfurilor primitivei, sau o culoare calculată în funcție de iluminare, anti-aliasing și texturare. Presupunând pentru moment culoarea constantă a unei primitive, aceasta se obține prin setarea unei variabile de stare a bibliotecii, variabila de culoare curentă (`GL_CURRENT_COLOR`). Culoarea curentă se setează folosind una din funcțiile `glColor#()`, care are mai multe variante, în funcție de tipul și numărul argumentelor. De exemplu, câteva din prototipurile acestei funcții definite în fișierul `gl.h` sunt:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
void glColor3ub(GLubyte r, GLubyte g, GLubyte b);
void glColor4d(GLdouble r, GLdouble g,
               GLdouble b, GLdouble a);
```

Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor roșu (r), verde (g), albastru (b), respectiv transparență (a) ca a patra

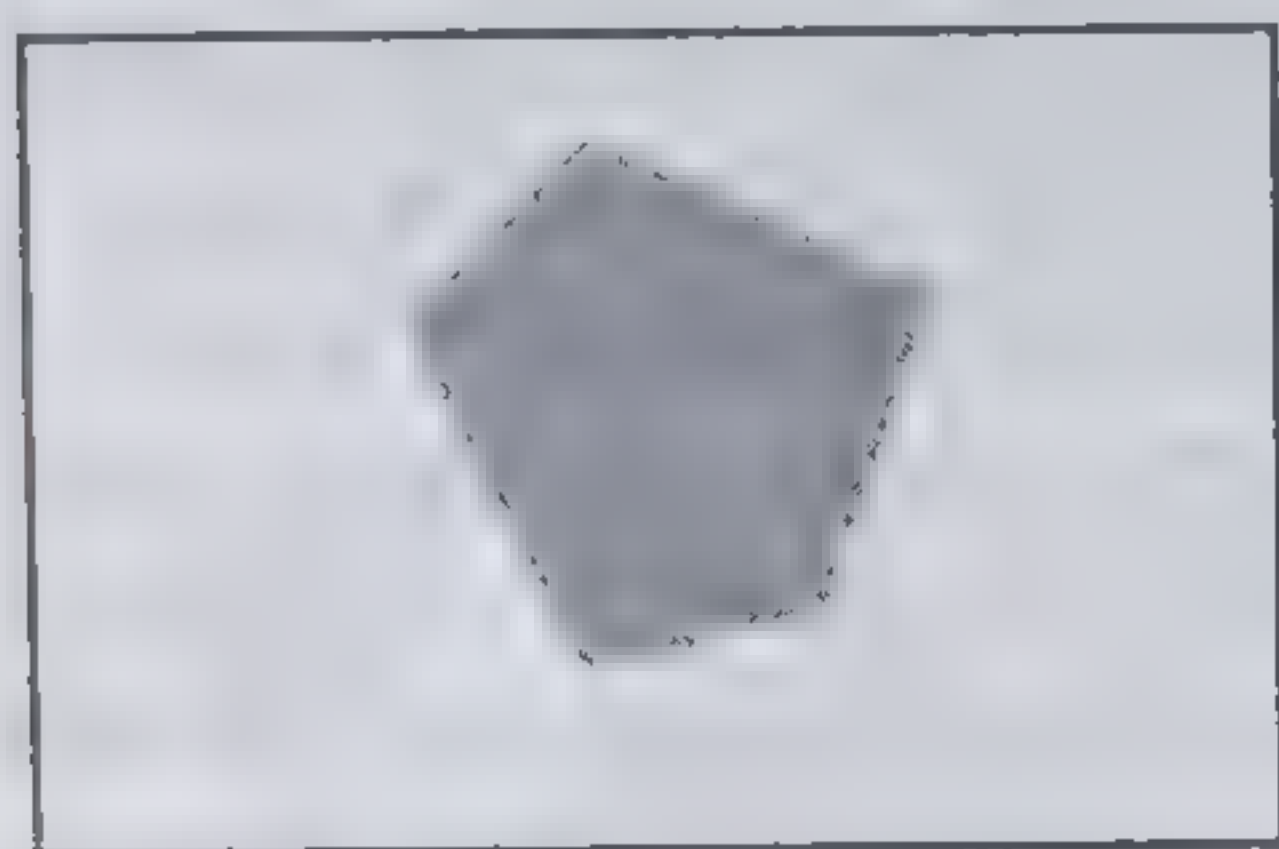


componentă pentru funcțiile cu 4 argumente. Fiecare componentă a culorii curente este memorată ca un număr în virgulă flotantă cuprins în intervalul  $[0,1]$ . Valorile argumentelor de tip întreg fără semn (unsigned int, unsigned char, etc.) sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea 0 este transformată în 0.0, iar valoarea maximă reprezentabilă este transformată în 1.0 (intensitate maximă). Valorile argumentelor de tip întreg cu semn sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea negativă maximă este transformată în -1.0, iar valoarea pozitivă maximă este transformată în 1.0 (intensitate maximă).

Valoarea finală a culorii unui pixel, rezultată din toate calculele de umbrire, anti-aliasing, texturare, etc, este memorată în bufferul de imagine, o componentă fiind reprezentată printr-un număr  $n$  de biți (nu neapărat același număr de biți pentru fiecare componentă). Deci componentele culorilor pixelilor memorate în bufferul de imagine sunt numere întregi în intervalul  $(0, 2^n - 1)$ , care se obțin prin conversia numerelor în virgulă flotantă prin care sunt reprezentate și prelucrate culorile primitivelor geometrice, ale materialelor, etc.

### ■ Exemplul 6.1

Secvența de funcții pentru redarea unui poligon în spațiul tridimensional și imaginea corespunzătoare generată de OpenGL arată în felul următor:



```
glColor3f(0.8,0.8,0.8);
glBegin(GL_POLYGON);
    glVertex3d(-4,2,-20);
    glVertex3d(-1,5,-20);
    glVertex3d(5,2.5,-20);
    glVertex3d(3,-3,-20);
    glVertex3d(-1,-4,-20);
glEnd();
```

Indentarea instrucțiunilor cuprinse între `glBegin()` și `glEnd()` nu este necesară în limbajul C în care este scris codul, ea este folosită doar pentru a evidenția începutul și sfârșitul primitivei geometrice.

În mod implicit, calculul transparenței nu este validat. Apelul funcției `glEnable(GL_BLEND)` validează combinarea culorilor, prin care culoarea fiecărui pixel al poligonului este combinată cu culoarea pixelului existent în bufferul de imagine. Modul în care are loc combinarea se selectează prin funcția `glBlendFunc()`. Prin combinarea culorilor se simulează transparența suprafețelor precum și alte efecte de redare a imaginilor (anti-aliasing, ceață). Exemple de transparență prin combinarea culorilor sunt date în §7.4.5.1.

### ■ Exemplul 6.2

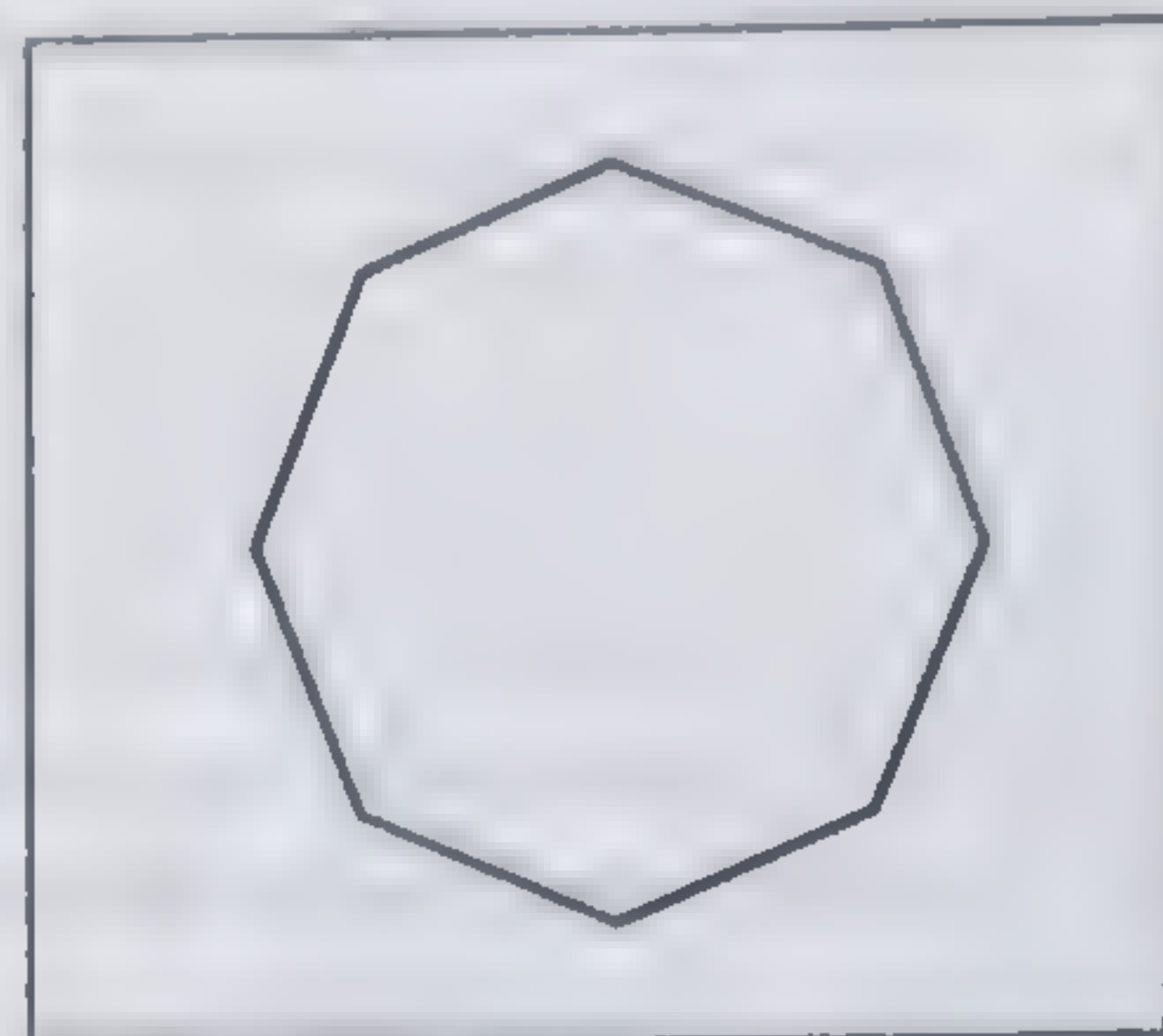
Se poate genera un poligon regulat prin calculul coordonatelor vârfurilor și folosirea lor ca argumente ale funcțiilor `glVertex#()`. De exemplu, calculul și desenarea în OpenGL a conturului unui octogon în spațiu este următoarea:



```

#define PI 3.141592
int n = 8;
double radius = 10;
glColor3d(0,0,0);
glBegin(GL_LINE_LOOP);
    for (int i=0;i<n;i++){
        double angle = 2*PI*i/n;
        glVertex3d(radius*cos(angle),
                    radius*sin(angle),-20);
    }
glEnd();

```



## 6.3 SISTEMUL DE VIZUALIZARE OPENGL

Sistemul de vizualizare OpenGL definește sistemele de referință, transformările geometrice și relațiile (matriceale) de transformare pentru redarea primitivelor geometrice. Sistemul de vizualizare OpenGL este o particularizare a sistemului PHIGS, în care centrul sistemului de referință de observare (VRP) coincide cu centrul de proiecție (PRP).

### 6.3.1 SISTEMELE DE REFERINȚĂ

Sistemul de referință universal definit în OpenGL este un sistem drept, iar matricea care reprezintă un punct (în plan, în spațiul tridimensional sau în coordonate omogene) este matrice coloană.

Pentru reprezentarea unei matrice 4x4 folosite pentru transformări geometrice în coordonate omogene, în OpenGL este utilizată convenția coloană majoră: fiind dat un număr de 16 valori reale (tip float sau double), în ordinea  $a_0, a_1, a_2, \dots, a_{15}$ , matricea corespunzătoare este:

$$A = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Aceste convenții sunt aceleași cu convențiile adoptate în lucrarea de față, deci matricele de transformări geometrice ale bibliotecii OpenGL sunt identice cu cele prezentate în secțiunile precedente.

Sistemul de referință de observare este un sistem de referință drept, definit ca localizare și orientare relativ la sistemul de referință universal. În acest sistem de referință de observare se poate defini o transformare de proiecție paralelă ortografică sau de proiecție perspectivă.

Funcțiile oferite de OpenGL permit definirea proiecției perspective pe un plan perpendicular pe axa z a sistemului de referință de observare, cu direcția de



observare spre  $-z$  (fig. 6.3). Trunchiul de piramidă de vizualizare este orientat spre  $-z_v$  și poate fi definit prin valorile:

- $z_{near}$ : distanța (dată ca valoare pozitivă) a planului de vizualizare și a planului de vizibilitate apropiat față de centrul sistemului de referință de observare ( $O_v$ )
- $z_{far}$ : distanța (dată ca valoare pozitivă) a planului de vizibilitate depărtat față de centrul sistemului de referință de observare
- $left, right$ : specifică coordonatele planelor verticale ale volumului de vizualizare
- $top, bottom$ : specifică coordonatele planelor orizontale ale volumului de vizualizare.

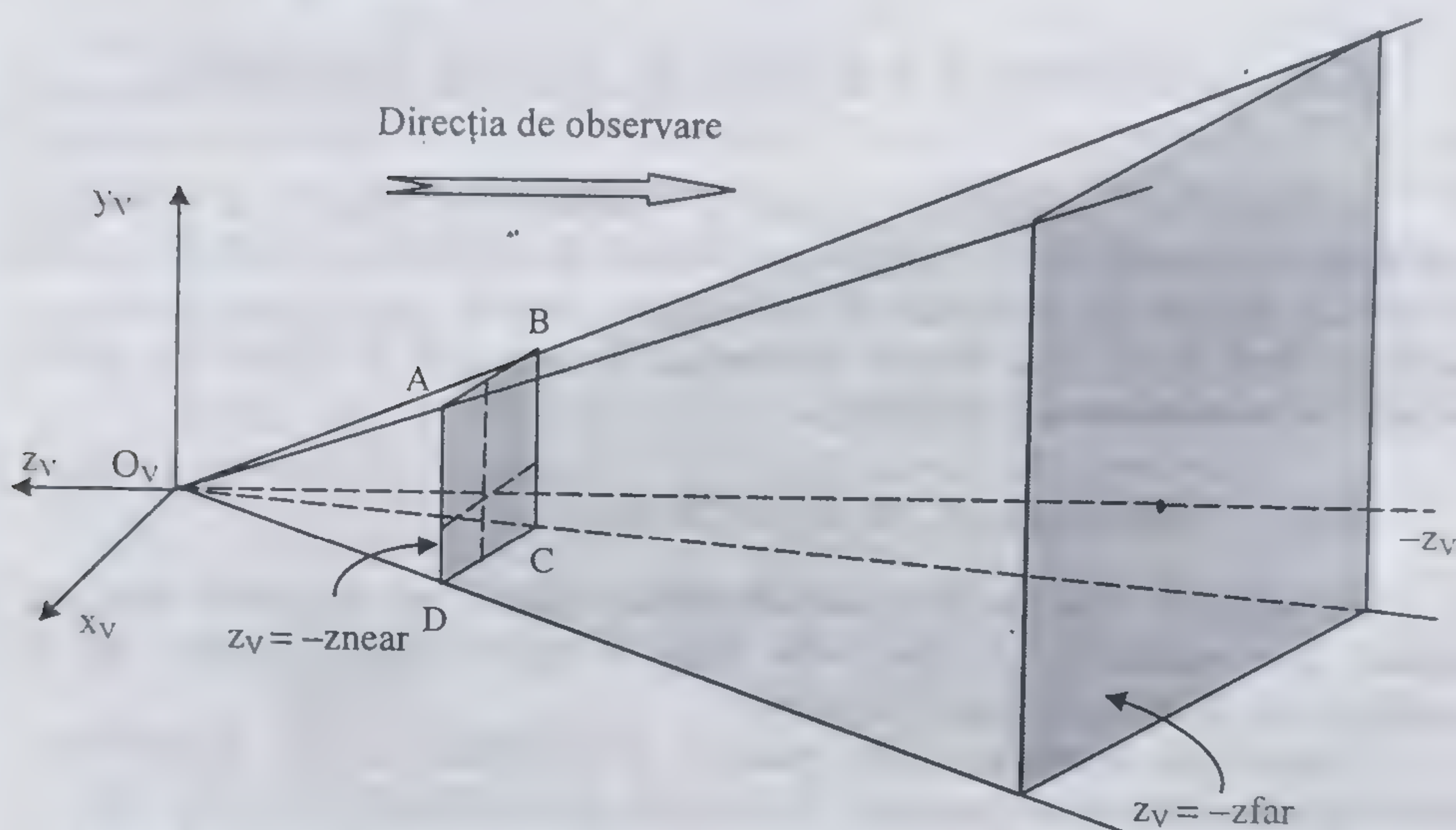


Fig. 6.3 Definirea proiecției perspective și a volumului de vizualizare în OpenGL.

Colțul din stânga-jos (D) al ferestrei de vizualizare are coordonatele ( $left, bottom, -z_{near}$ ) în sistemul de referință de observare, iar colțul din dreapta-sus (B) are coordonatele ( $right, top, -z_{near}$ ) în același sistem.

Funcția OpenGL care definește o astfel de proiecție perspectivă este funcția `glFrustum()` care are prototipul:

```
void glFrustum(GLdouble left, GLdouble right,
               GLdouble bottom, GLdouble top,
               GLdouble znear, GLdouble zfar);
```

Matricea de normalizare calculată în acest caz este o particularizare a produsului matricelor date de relațiile (4.14), (4.15), (4.16), în care:  $n = -z_{near}$ ;  $d = -z_{near}$ ;  $f = -z_{far}$ ;  $x_{min} = left$ ;  $x_{max} = right$ ;  $y_{min} = bottom$ ;  $y_{max} = top$ . Deducerea ei este propusă ca un exercițiu pentru cititori.



### 6.3.2 TRANSFORMĂRI GEOMETRICE

Așa după cum s-a mai arătat, nu este eficient să se calculeze produsul dintre matricea de reprezentare a fiecărui punct și matricele de transformări succesive, ci se calculează o matrice de transformare compusă, care se poate aplica unuia sau mai multor obiecte. Pentru calculul matricelor de transformare compuse (prin produs de matrice) se folosește o *matrice de transformare curentă* și operații de acumulare în matricea curentă prin postmultiplicare (înmulțire la dreapta): se înmulțește matricea curentă cu noua matrice (în această ordine) și rezultatul înlocuiește conținutul matricei curente.

Pentru secvența de transformări în ordinea  $M_1, M_2 \dots M_n$ , aplicate obiectelor, se inițializează matricea curentă  $C$  cu matricea identitate, după care se calculează matricea compusă prin postmultiplicare în ordine inversă celei în care se aplică matricele componente:

$$C = I$$

$$C = CM_n = M_n$$

$$C = CM_{n-1} = M_n M_{n-1}$$

.....

$$C = CM_2 = M_n M_{n-1} \dots M_2$$

$$C = CM_1 = M_n M_{n-1} \dots M_2 M_1$$

$$C = M = M_n M_{n-1} \dots M_2 M_1$$

$$P' = MP = M_n M_{n-1} \dots M_2 M_1 P$$

În această transformare, se aplică mai întâi matricea  $M_1$  punctului  $P$ ; punctului transformat rezultat  $i$  se aplică transformarea  $M_2$ , ș.a.m.d. Se poate reține deci că matricea compusă  $M$  se scrie ca produs în ordine inversă aplicării transformărilor componente și tot în ordine inversă se acumulează în matricea curentă. Acest mod de calcul al matricelor compuse are suport în OpenGL prin mai multe variabile interne (matrice curente) și funcții prin care acestea pot fi calculate. Pentru început, se consideră o matrice curentă oarecare  $C$  stabilită în OpenGL printr-o comandă corespunzătoare (va fi prezentată ulterior).

Matricea curentă se poate inițializa cu matricea identitate prin funcția `glLoadIdentity()` sau cu o matrice oarecare, dată prin pointer la un vector de 16 valori consecutive, prin funcția `glLoadMatrix#()`.

```
glLoadMatrixd(const GLdouble* m);
glLoadMatrixf(const GLfloat* m);
```

Valorile din vectorul `GLdouble* m` (respectiv `GLfloat* m`) sunt atribuite în ordinea coloană majoră matricei curente.

Conținutul matricei curente poate fi modificat prin multiplicare (la dreapta) cu o altă matrice, dată printr-un vector de 16 valori de tip `double` sau `float` utilizând funcția `glMultMatrix#()`.



```
glMultMatrixd(const GLdouble* m);
glMultMatrixf(const GLfloat* m);
```

Matricea curentă  $C$  este înlocuită cu produsul  $C M$ , unde  $M$  este matricea corespunzătoare vectorului dat prin pointerul  $m$ . Crearea unei matrice pentru o transformare elementară (translație, scalare, etc) și înmulțirea ei cu matricea curentă se poate face prin apelul unei singure funcții OpenGL.

**Transformarea de translație** cu valorile  $x, y, z$  se implementează prin apelul unuia din funcțiile `glTranslated()` sau `glTranslatef()`, după tipul argumentelor de apel:

```
glTranslated(GLdouble x, GLdouble y, GLdouble z);
glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glTranslate#()` creează o matrice de translație  $T(x, y, z)$ , dată de relația (3.2), înmulțește la dreapta matricea curentă  $C$ , iar rezultatul înlocuiește conținutul matricei curente  $C$ , deci:  $C = C T(x, y, z)$ .

**Transformarea de scalare** este efectuată de una din funcțiile `glScaled()` sau `glScalef()`.

```
glScaled(GLdouble x, GLdouble y, GLdouble z);
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glScale#()` creează o matrice de scalare  $S(x, y, z)$  și o înmulțește cu matricea curentă, rezultatul fiind depus în matricea curentă.

**Transformarea de rotație** se definește printr-o direcție de rotație dată prin vectorul de poziție  $x, y, z$  și un unghi `angle` (specificat în grade). Prototipurile funcțiilor de rotație sunt:

```
glRotated(GLdouble angle, GLdouble x,
           GLdouble y, GLdouble z);
glRotatef(GLfloat angle, GLfloat x,
           GLfloat y, GLfloat z);
```

Rotațiile în raport cu axele de coordonate sunt cazuri particulare ale funcțiilor `glRotate#()`. De exemplu, rotația cu unghiul `angle` în raport cu axa  $x$  se obține la apelul funcției `glRotated(angle, 1, 0, 0)`.

**Transformările compuse** se efectuează prin acumularea produsului matricelor componente în matricea curentă. De exemplu, transformările de modelare din exemplul 3.2 au fost implementate în OpenGL după cum urmează.

```
Transformarea  $M_2 = T(0, -8, 0) S(2, 2, 2)$ :
glTranslated(0, -8, 0);
glScaled(2, 2, 2);
```

```
Transformarea  $M_3 = T(8, 0, 0) R_z(\pi/4) S(2, 1, 2)$ :
glTranslated(8, 0, 0);
glRotated(45, 0, 0, 1);
glScaled(2, 1, 2);
```



Transformarea  $M_4 = T(-8,0,0)R_z(-\pi/4)S(2,1,2)$ :

```
glTranslated(-8, 0, 0);
glRotated(-45, 0, 0, 1);
glScaled(2, 1, 2);
```

**Stivele matricelor de transformare.** După cum s-a mai arătat, transformarea unui obiect din sistemul de referință local (de modelare) în sistemul de referință normalizat este compusă din succesiunea transformărilor de modelare (instanțiere)  $M_I$ , de observare  $M_V$  și de normalizare  $M_N$ :

$$M = M_N M_V M_I$$

Transformarea de instanțiere  $M_I$  este, în general, specifică fiecărui obiect, deci se calculează pentru fiecare obiect în parte.

Pentru cele mai multe din aplicațiile grafice, în care scena este observată dintr-un singur punct de observare, matricea de observare  $M_V$  este unică pentru toate obiectele într-un cadru dat, dar se modifică la cadrul următor (imaginea următoare), dacă observatorul și-a schimbat poziția de observare, situație cea mai probabilă în grafica interactivă.

Matricea de normalizare  $M_N$  este definită de parametrii de proiecție, care, așa cum s-a prezentat în secțiunea precedentă, corespund construcției sistemului grafic (tipul și numărul de proiectoare, unghiul de vizibilitate), deci această matrice este o caracteristică constructivă a sistemului grafic și rămîne constantă pe toată perioada desfășurării programului, pentru toate obiectele și pentru toate imaginile generate (cadrele de imagine). În concluzie, sunt necesare următoarele matrice de transformare în cursul generării imaginilor succesive, la momentele (cadrele)  $i$ ,  $i+1$ ,  $i+2$ , ...

Cadrul  $i$ :  $M = M_N M_{V_i} M_{I1}$ , pentru obiectul 1  
 $M = M_N M_{V_i} M_{I2}$ , pentru obiectul 2

.....  
 $M = M_N M_{V_i} M_{Ik}$ , pentru obiectul  $k$

Cadrul  $i+1$ :  $M = M_N M_{V_{i+1}} M_{I1}$ , pentru obiectul 1  
 $M = M_N M_{V_{i+1}} M_{I2}$ , pentru obiectul 2

.....  
 $M = M_N M_{V_{i+1}} M_{Ik}$ , pentru obiectul  $k$

Se pune problema care este cea mai eficientă organizare a acestor operații de transformări succesive. Dacă s-ar utiliza o singură matrice curentă, secvența operațiilor de compunere ar trebui să fie efectuată pentru fiecare obiect, în fiecare cadru de imagine. Este evident că utilizarea unei singure matrice curente de transformare este inefficientă și nejustificată, dat fiind că o matrice este o resursă extrem de puțin costisitoare. Soluția adoptată în OpenGL este de a se folosi mai multe stive de matrice de transformare, în care să fie reținute și reutilizate matricele intermediare de transformare.

Transformările de modelare și de observare sunt efectuate într-o stivă de matrice, numită stiva matricelor de modelare-vizualizare (*modelview matrix stack*).



Transformarea de normalizare este prelucrată într-o stivă de matrice separată, numită stiva matricelor de proiecție (*projection matrix stack*). Mai există încă o stivă de matrice pentru operațiile de texturare, numită stiva matricelor de texturare (*texture matrix stack*). Separarea matricelor de modelare vizualizare și de proiecție permite execuția în paralel a operațiilor din pipeline-ul grafic al acceleratoarelor hardware care au prevăzute resurse pentru astfel de prelucrări.

Fiecare stivă de matrice se comportă asemănător cu o stivă obișnuită, asupra căreia se operează prin funcții de introducere (*push*) și extragere (*pop*). Aceste funcții diferă foarte puțin față de funcțiile *push* și *pop* folosite în general în programare.

În orice moment, una din cele trei stive de matrice este declarată ca stivă curentă și toate operațiile cu matricele de transformare sunt adresate stivei curente. Setarea unei stive curente se face prin apelul funcției

```
void glMatrixMode(GLenum mode);
```

unde argumentul *mode* poate lua una din constantele simbolice *GL\_MODELVIEW*, *GL\_PROJECTION* sau *GL\_TEXTURE* pentru setarea ca stivă curentă a stivei de modelare-vizualizare, a stivei de proiecție sau a stivei de texturare

Matricea din vârful stivei curente este matricea curentă *C*, cu care se efectuează operațiile cu matrice (*glLoadIdentity()*, *glTranslate#()*, *glLoadMatrix#()*, *glMultMatrix#()*; etc).

Funcția:

```
void glPushMatrix();
```

adaugă o nouă matrice în stiva curentă, egală cu matricea din vârful acesteia, care devine noul vârf al stivei (fig. 6.4). Încărcarea unei noi valori în noua matrice curentă se face prin operații ulterioare de încărcare sau acumulare prin postmultiplicare. De fapt, aceasta este caracteristica cea mai importantă a stivelor matricelor de transformări, și anume că operația *push* se execută prin două sau mai multe funcții: *glPushMatrix()* creează o nouă poziție în capul stivei, identică cu valoarea precedentă, și în această poziție se continuă operațiile de compunere a matricelor prin postmultiplicare.

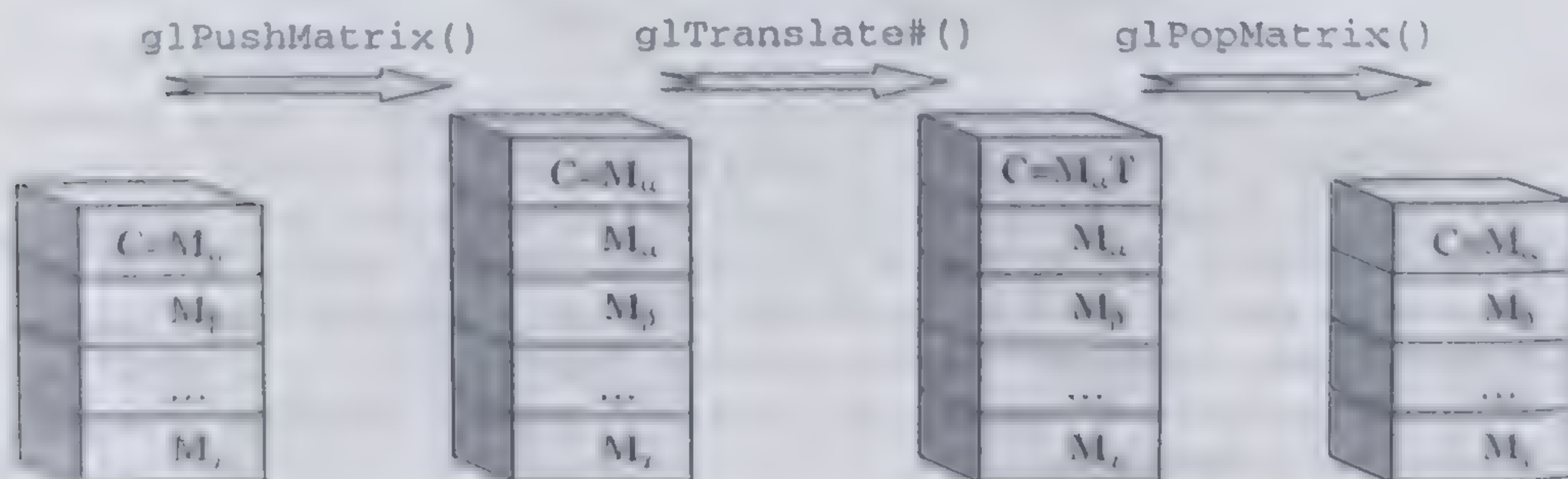


Fig. 6.4 Operații cu stiva matricelor de transformare.



Funcția:

```
void glPopMatrix();
```

elimină matricea din vârful stivei curente, iar matricea precedentă acesteia devine noua matrice curentă. Această funcție nu salvează matricea eliminată (deci ea nu este "extrasă", ca în operațiile pop obișnuite), iar dacă această matrice ar mai fi necesară, ea trebuie copiată în altă zonă de memorie, înainte de apelul funcției `glPopMatrix()`.

Funcția `glPushMatrix()` se apelează atunci când o matrice aflată în capul stivei trebuie salvată pentru a fi utilizată ulterior. Funcția `glPopMatrix()` se apelează atunci când operațiile de compunere a matricelor trebuie să continue cu o matrice aflată *sub* capul stivei. La inițializarea bibliotecii OpenGL, fiecare stivă de matrice conține o matrice care este vârful stivei respective.

Se poate prezenta acum un program OpenGL simplu, folosind utilitarul GLUT, care exemplifică folosirea stivelor matricelor de transformări geometrice.

### ■ Exemplul 6.3

Se reia ca exemplu de utilizare a stivelor matricelor de transformare programul din exemplul 4.2, în care se generează imaginea unei scene obținute prin instanțierea a patru obiecte.

Matricea de normalizare și proiecție se calculează o singură dată și se introduce în vârful stivei matricelor de proiecție. Pentru aceasta, se setează ca stivă curentă stiva matricelor de proiecție și se inițializează matricea din vârful acestei stive cu matricea identitate:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

Valoarea matricei de normalizare se setează folosind funcția `glFrustum()` descrisă anterior. Funcția `glFrustum()` creează matricea de normalizare corespunzătoare argumentelor de apel, o înmulțește cu matricea curentă (care are valoarea de matrice identitate) și depune rezultatul în matricea curentă, care este matricea din vârful stivel matricelor de proiecție. În acest exemplu se apelează:

```
glFrustum(-1, 1, -1, 1, 1, 40);
```

Matricele de transformare de modelare și observare se compun în stiva matricelor de modelare-vizualizare. Sistemul de referință de observare are centrul în punctul  $(x_v, y_v, z_v)$  și este rotit cu un unghi  $\gamma$  în raport cu axa  $z$ . Pașii de calcul al matricelor sunt următorii:

- (1) Mai întâi se setează ca stivă curentă stiva de modelare-vizualizare și se inițializează matricea din vârful acesteia cu matricea identitate:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

- (2) Se creează în stiva curentă matricea de transformare de observare  $M_v = R_z(-\gamma)T(-x_v, -y_v, -z_v)$ :



```
glRotated(-gama, 0, 0, 1);
glTranslated(-xv, -zv, -zv);
```

- (3) Primul obiect are matricea de instanțiere  $M_1 = I$ , deci matricea curentă (care conține matricea de vizualizare) se aplică coordonatelor vârfurilor cubului din sistemul de referință de modelare. Funcția `Cube()` transmite aceste vârfuri către OpenGL.
- (4) Dacă s-ar construi în acest moment matricea de transformare pentru obiectul al doilea, prin multiplicarea matricei curente cu matricele componente de instanțiere din  $M_2 = T(0, -8, 0) S(2, 2, 2)$ , atunci matricea de observare s-ar pierde și pentru următorul obiect ar trebui calculată din nou. De aceea, se reține în stivă matricea de observare prin apelul funcției `glPushMatrix()`, care creează o nouă matrice în vârful stivei, cu aceeași valoare (egală cu  $M_v$ ). Matricea de instanțiere  $M_2$  se compune cu matricea de observare în această nouă matrice curentă, se apelează funcția `Cube()`, după care matricea din vârful stivei ( $M_v M_2$ ) nu mai este necesară. Prin apelul funcției `glPopMatrix()`, această matrice este eliminată și, în același timp, se regăsește în vârful stivei matricea de observare  $M_v$ , care va putea fi folosită pentru obiectele următoare. Funcțiile apelate în acest pas sunt:

```
glPushMatrix();
    glTranslated(0, -8, 0);
    glScaled(2, 2, 2);
    Cube();
glPopMatrix();
```

- (5) Pentru celelalte obiecte se repetă operațiile de la pasul 4.

Programul următor, care conține cele explicate mai sus, este un exemplu tipic de program grafic folosind biblioteca OpenGL și utilitarul GLUT. În acest program se poate urmări inițializarea bibliotecii GLUT, definirea funcțiilor `callback` și modul de utilizare a stivelor de transformări geometrice. Unele funcții nou introduse în program sunt explicate în comentarii.

```
#include <gl/glut.h>
// Initializari
void Init(){
    glClearColor(1.0, 1.0, 1.0, 0.0); // culoare stergere
    glEnable(GL_DEPTH_TEST);          // validare Z-buffer
}
// Redare cub
void Cube(){
    glColor3f(0.8, 0.8, 0.8);
    glBegin(GL_POLYGON);
        glVertex3d(-1, -1, 1);
        glVertex3d( 1, -1, 1);
        glVertex3d( 1,  1, 1);
        glVertex3d(-1,  1, 1);
    glEnd();
```



```
glColor3f(0,0,0);
glBegin(GL_LINE_LOOP);
    glVertex3d(-1,-1,-1);
    glVertex3d( 1,-1,-1);
    glVertex3d( 1, 1,-1);
    glVertex3d(-1, 1,-1);
glEnd();
glBegin(GL_LINE_LOOP);
    glVertex3d(-1,-1,-1);
    glVertex3d( 1,-1,-1);
    glVertex3d( 1,-1, 1);
    glVertex3d(-1,-1, 1);
glEnd();
glBegin(GL_LINE_LOOP);
    glVertex3d(-1,1,-1);
    glVertex3d( 1,1,-1);
    glVertex3d( 1,1, 1);
    glVertex3d(-1,1, 1);
glEnd();
glBegin(GL_LINE_LOOP);
    glVertex3d(-1,-1,-1);
    glVertex3d(-1, 1,-1);
    glVertex3d(-1, 1, 1);
    glVertex3d(-1,-1, 1);
glEnd();
glBegin(GL_LINE_LOOP);
    glVertex3d(1,-1,-1);
    glVertex3d(1, 1,-1);
    glVertex3d(1, 1, 1);
    glVertex3d(1,-1, 1);
glEnd();

// Generare axe de coordonate
void Axes(double x,double y, double z){
    glColor3f(0,0,0);
    glBegin(GL_LINES);
        glVertex3d(0,0,0);
        glVertex3d(x,0,0);
        glVertex3d(0,0,0);
        glVertex3d(0,y,0);
        glVertex3d(0,0,0);
        glVertex3d(0,0,z);
    glEnd();
    // Texte x, y, z
    glColor3ub(0, 0, 0);
    glRasterPos3f(1.1*x, 0.0, 0.0);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'x');
    glRasterPos3f(0.0, 1.1*y, 0.0);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'y');
    glRasterPos3f(0.0, 0.0, 1.1*z);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'z');
```



```

// SAGETI
.....
}
// Functia apelata la schimbarea dim. ferestrei
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-(GLfloat)w/(GLfloat)h,
              (GLfloat)w/(GLfloat)h, -1, 1, 1, 40);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
static double xv = 2, yv = 2, zv = 20;
static double gama = 0;

// Functia callback de generare imagine
void Display(){
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotated(-gama, 0,0,1);
    glTranslated(-2,-2,-20);
    // transf. de obs.:  $C = R_z(-30)T(-2,-2,-20)$ 

    Axes(16,8,16);
    Cube(); // primul obiect

    glPushMatrix(); // al doilea obiect
    glTranslated(0,-8,0);
    glScaled(2,2,2);
    //  $C = R_z(-30)T(-2,-2,-20)T(0,-8,0)S(2,2,2)$ 
    Cube();
    glPopMatrix(); //  $C = R_z(-30)T(-2,-2,-20)$ 
    glPushMatrix(); // al treilea obiect
    glTranslated(8,0,0);
    glRotated(45,0,0,1);
    glScaled(2,1,2);
    //  $C = R_z(-30)T(-2,-2,-20)T(8,0,0)R_z(45)S(2,2,2)$ 
    Cube();
    glPopMatrix(); //  $C = R_z(-30)T(-2,-2,-20)$ 
    glPushMatrix(); // al patrulea obiect
    glTranslated(-8,0,0);
    glRotated(-45,0,0,1);
    glScaled(2,1,2);
    //  $C = R_z(-30)T(-2,-2,-20)T(-8,0,0)R_z(-45)S(2,1,2)$ 
    Cube();
    glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();
}

```



```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Transformări grafice");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

În funcția locală de inițializare `Init()` se setează culoarea de ștergere a bufferului de imagine folosind funcția `glClearColor()`. Această culoare de ștergere este memorată într-o variabilă de stare și este folosită pentru ștergerea bufferului de imagine atunci când se apelează funcția `glClear()`. De asemenea, se validează testul de adâncime, pentru eliminarea suprafețelor ascunse.

Funcția cea mai importantă a programului este funcția `Display()` în care se execută toate operațiile de generare a unei imagini: ștergerea bufferului de imagine, (funcția `glClear()`), crearea matricelor de transformare în stiva matricelor de modelare-vizualizare, generarea imaginii a patru obiecte aplicând transformarea corespunzătoare modelului cub (funcția `Cube()`).

La sfârșitul operațiilor de generare a imaginii, se comută bufferul de imagine prin apelul funcției `glutSwapBuffers()`. Valoarea matricei curente în fiecare pas de execuție este scrisă în comentarii. Indentările instrucțiunilor nu sunt necesare, dar permit urmărirea mai ușoară a operațiilor cu stivele de matrice. Funcția callback `Display` este apelată de sistemul de operare ori de câte ori este necesară redesenarea ferestrei de afișare.

O formă foarte simplificată de desenare a unui obiect tridimensional format din mai multe fețe, fiecare față formată din mai multe vârfuri, este dată în funcția `Cube()`. Către OpenGL se transmit primitive geometrice compuse din vârfurile fiecărei fețe, date în coordonate în sistemul de referință local (de modelare). Dacă se urmărește diagrama de execuție a operațiilor în OpenGL din fig. 6.1, se vede că asupra vârfurilor unei primitive se aplică transformarea din matricea curentă din stiva de modelare-vizualizare. De aceea este posibil ca, pentru fiecare instanțiere, să fie transmise aceleași coordonate (ale modelului), care sunt transformate (instanțiere și observare) în sistemul de referință de observare. În exemplul dat, prima față se desenează ca un poligon plin de culoare gri, iar celelalte, ca muchii de culoare neagră, pentru o mai bună percepere a imaginii desenate alb-negru.

Secvența de operații continuă cu: asamblarea primitivelor geometrice, folosind informațiile topologice transmise de funcția `glBegin()` și vârfurile primitivei transformate în sistemul de referință de observare, normalizarea, folosind matricea din vârful stivei matricelor de proiecție, decuparea, împărțirea



omogenă, proiecția, transformarea fereastră poartă și conversia de rastru pentru înscriserea valorilor corespunzătoare a pixelilor primitivei geometrice.

Funcția de redare a axelor de coordonate `Axes()` generează axele de coordonate și numele lor, printr-un caracter afișat folosind funcția `glutBitmapCharacter()`. Partea de desenare a săgeților axelor nu este prezentată, dar se poate completa cu ușurință.

Funcția callback de redimensionare `Reshape()` primește la apelul ei de către sistem dimensiunile  $w$  și  $h$  ale ferestrei de afișare. În această funcție se setează dimensiunea porții egală cu dimensiunea ferestrei de afișare și cu colțul stânga-jos în punctul de coordonate  $(0,0)$  al ferestrei, prin apelul funcției `glViewport()`. Tot în această funcție callback este definită transformarea de normalizare și proiecție perspectivă (`glFrustum()`) astfel încât scalarea dintre fereastra de vizualizare și poarta de afișare (care a fost setată egală cu fereastra de afișare) să fie uniformă, adică raportul dintre dimensiunile pe orizontală și verticală ale ferestrei de vizualizare să fie egal cu raportul dintre dimensiunile pe orizontală și verticală ale porții de afișare. Funcția de definire a proiecției perspectivă și a volumului de vizualizare, `glFrustum()` operează asupra stivei matricelor de proiecție. Funcția `Reshape()` este apelată atât la lansarea programului aplicației, cât și la orice modificare a dimensiunilor ferestrei.

În funcția `main()` a programului aplicației sunt apelate funcțiile GLUT de inițializare a ferestrei de afișare, sunt înregistrate funcțiile callback necesare și apoi se intră în execuția programului la apelul funcției `glutMainLoop()`.

Imaginile din fig. 4.7 a,b,c,d,e,f s-au obținut prin modificarea corespunzătoare a valorilor variabilelor  $xv$ ,  $yv$ ,  $zv$ ,  $\gamma$  și recompilarea programului.

Stivele matricelor de transformare sunt deosebit de utile pentru redarea scenelor și a obiectelor modelate ierarhic, în care scena sau fiecare obiect este compus din mai multe subobiecte, pe mai multe nivele de ierarhie. Fiecare obiect component este definit printr-o matrice de localizare relativ la obiectul căruia îi aparține (obiect părinte), și instanțierea acestuia se obține prin compunerea matricii de localizare proprii cu matricea de localizare a obiectului părinte. Reprezentarea și redarea scenelor modelate ierarhic este tratată în cap. 11, dar în exemplul următor se va prezenta un scurt program de aplicație care folosește compunerea mai multor transformări de modelare.

#### ■ Exemplul 6.4

Programul descrie construirea unui sistem solar foarte simplificat, cu soarele, două planete și un satelit al uneia dintre planete. Sferele care reprezintă aceste corpuri sunt redată folosind funcțiile `glutSolidSphere()` și `glutWireSphere()` care primesc ca argumente dimensiunea, numărul de paralele și numărul de meridiane ale sferei.

Se consideră că soarele se află în centrul sistemului de referință universal și că planetele execută o mișcare de rotație în jurul soarelui în planul  $z = 0$ , la distanțe diferite. Satelitul execută o mișcare de rotație în jurul unei planete, de asemenea în



planul  $z = 0$ . Sistemul de referință de observare are axele paralele cu axele sistemului universal și centrul pe axa  $z$  pozitivă, la o valoare convenabil aleasă pentru a avea obiectele în câmpul de vizibilitate.

Poziția centrului unei planete depinde de raza de rotație  $r_p$  în jurul soarelui și de unghiul de rotație (dat de variabila  $\rho_p$  care variază între  $0$  și  $360^\circ$ ) și are coordonatele  $C_p(r_p \cos \rho_p, r_p \sin \rho_p, 0)$ . Se poate observa ușor că aceste coordonate se obțin printr-o translație cu  $T(r_p, 0, 0)$ , urmată de o rotație cu  $R_z(\rho_p)$ , deci matricea de instanțiere a unei planete este  $M_p = R_z(\rho_p)T(r_p, 0, 0)$ .

Transformarea de instanțiere a satelitului se calculează relativ la sistemul de referință local al planetei sale. La fel ca și pentru o planetă, poziția centrului satelitului depinde de raza de rotație ( $r_s$ ) a acestuia și unghiul de rotație ( $\rho_s$ ), și are coordonatele în sistemul de referință local al planetei  $C_s(r_s \cos \rho_s, r_s \sin \rho_s, 0)$ . Rezultă că matricea de instanțiere a satelitului relativ la sistemul de referință al planetei sale este  $M_s = R_z(\rho_s)T(r_s, 0, 0)$ . Programul de reprezentare a acestui sistem solar este următorul:

```
#include <gl/glut.h>
#define rp1 1.8
#define rp2 1.2
#define rs1 0.6
static int rop1 = 20, rop2 = 40, ros1 = 20;
static float zpos = 5;
void Init(){
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glEnable(GL_DEPTH_TEST);
}
void Axes(double x, double y, double z){
    // aceeași funcție ca în exemplul precedent
}
void Keyboard(unsigned char key, int x, int y){
    switch (key){
        case 'Z':
            zpos = zpos + 0.1;
            glutPostRedisplay();
            break;
        case 'z':
            zpos = zpos - 0.1;
            glutPostRedisplay();
            break;
        case 'P':
            rop1 = (rop1 + 10) % 360;
            rop2 = (rop2 + 10) % 360;
            glutPostRedisplay();
            break;
        case 'p':
            rop1 = (rop1 - 10) % 360;
            rop2 = (rop2 - 10) % 360;
            glutPostRedisplay();
            break;
    }
}
```



```

        break;
    case 'S':
        ros1 = (ros1+10)%360;
        glutPostRedisplay();
        break;
    case 's':
        ros1 = (ros1-10)%360;
        glutPostRedisplay();
        break;
    }
}

void Display(void) {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
        glTranslatef(-1,-1,-zpos);          //transf.de obs.
        Axes(3,1.5,2.5);
        glColor3f(0.4,0.4,0.4);
        glutSolidSphere(0.6, 16, 16);      // soare

        glColor3f(0,0,0);
        glPushMatrix();
            glRotatef(rop1, 0,0,1);
            glTranslatef(rp1,0,0);
            glutWireSphere(0.25, 8, 8);    // planeta 1
            Axes(1,0.8,1.8);
            glRotatef(ros1, 0,0,1);
            glTranslatef(rs1,0,0);
            glutSolidSphere(0.1, 8, 8);    // satelit 1
            Axes(0.5,0.5,1);
        glPopMatrix();

        glRotatef(rop2, 0,0,1);
        glTranslatef(rp2,0,0);
        glutWireSphere(0.2, 8, 8);         // planeta 2
        Axes(0.5,1.1,2.5);
    glPopMatrix();
    glutSwapBuffers();
}

void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (GLfloat)w/(GLfloat)h, 1, 40);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);

```



```

    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Sistem Solar");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMainLoop();
    return 0;
}

```

Imaginile obținute la execuția acestui program sunt prezentate în fig. 6.5. În funcția `Init()` se validează funcționarea algoritmului de Z-buffer, pentru ascunderea suprafețelor ascunse în funcție de adâncime (coordonata  $z$  în sistemul de referință normalizat) prin funcția `glEnable(GL_DEPTH_TEST)`. Valoarea cu care este șters bufferul de adâncime la începutul fiecărui cadru (la execuția funcției `glClear()`) are implicit valoarea maximă a coordonatei  $z$  a volumului canonic în sistemul de referință normalizat (1.0).

În funcția `Reshape()` este utilizată o altă funcție de definire a proiecției perspectivă, funcția `gluPerspective()` din biblioteca GLU, care este o versiune simplificată a funcției `glFrustum()`. Prototipul acestei funcții este:

```

void gluPerspective(GLdouble fovy, GLdouble aspect,
                    GLdouble zNear, GLdouble zFar);

```

unde `fovy` este unghiul de vizibilitate pe verticală (în grade); `aspect` este raportul  $w/h$  a dimensiunilor pe orizontală și verticală a ferestrei de afișare; `zNear` și `zFar` sunt valorile absolute ale distanțelor planelor de vizibilitate apropiată, respectiv depărtată.

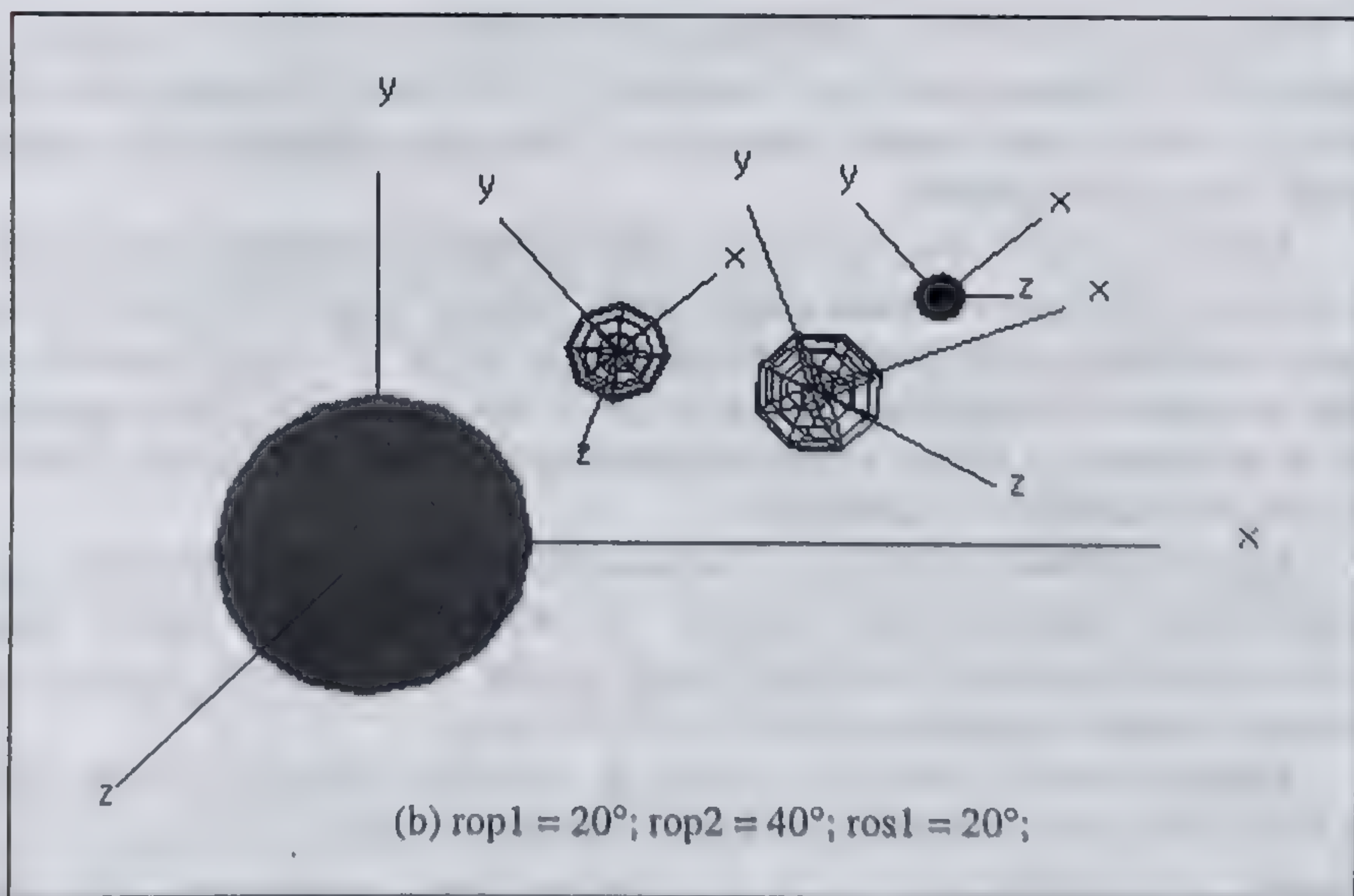
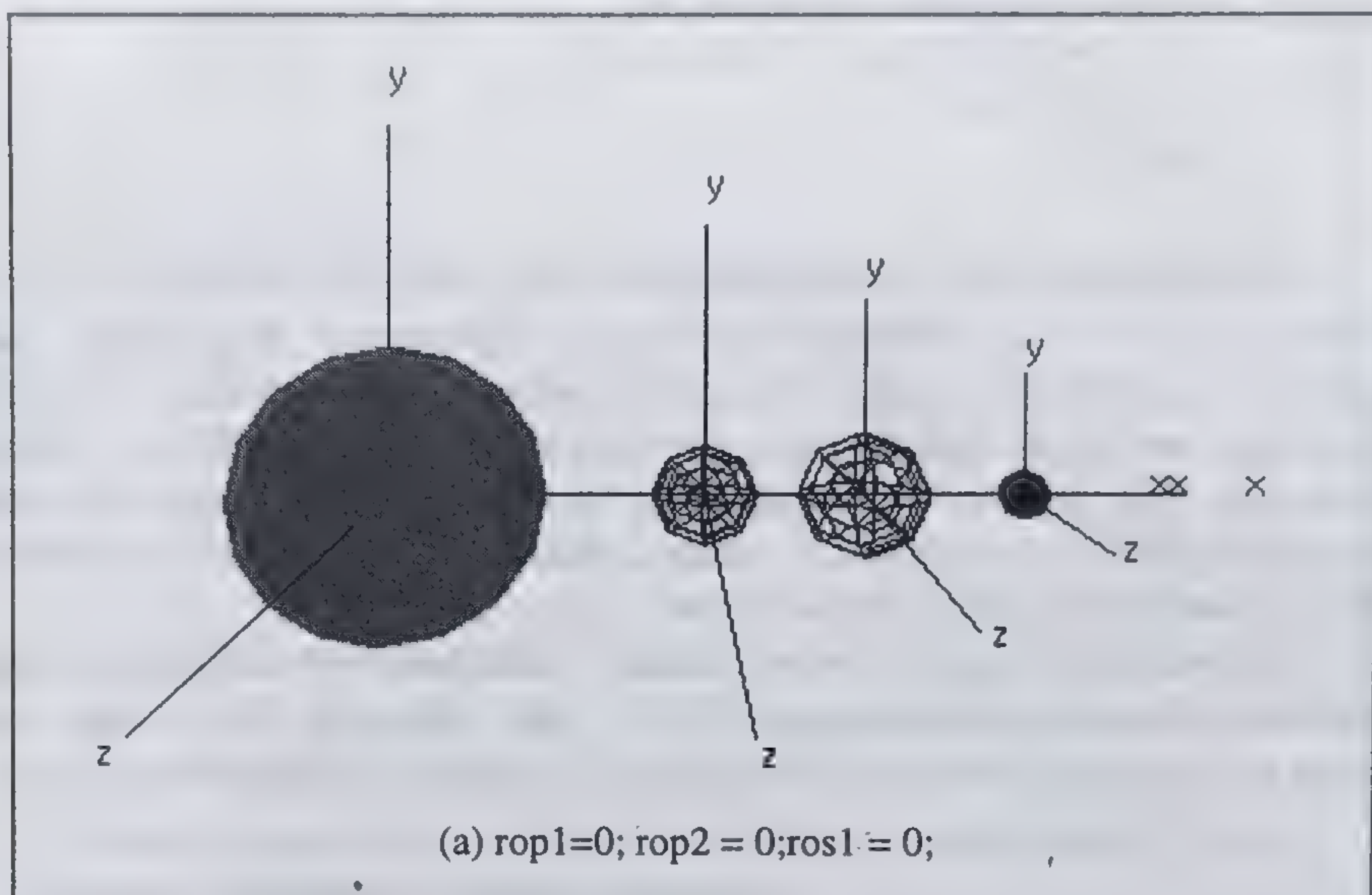
Funcția callback `Keyboard()` este apelată la acționarea unei taste. În acest exemplu simplu sunt tratate câteva taste. Tasta 'z' scade, iar tasta 'Z' crește valoarea coordonatei  $z_v$  a observatorului. Tastele 'p' și 'P' scad, respectiv, cresc unghiul de rotație al planetelor. Tastele 's' și 'S' scad, respectiv, cresc unghiul de rotație al satelitului. Se obține o mișcare interactivă (foarte simplificată, e adevărat, dar intuitivă) a planetelor și satelitului.

Funcția callback `Display()` se execută la fiecare redesenare care are loc la modificarea dimensiunilor ferestrei de afișare și prin apelul funcției `glutPostRedisplay()` la modificarea poziției de observare (`zpos`) sau a uneia dintre valorile unghiurilor `rop1`, `rop2` sau `ros1`.

Păstrarea matricei identitate în stiva de modelare-vizualizare, astfel încât să poată fi folosită pentru o nouă secvență de compunere se realizează prin perechea de funcții `glPushMatrix()` și `glPopMatrix()` exterioară din funcția `Display()`. Matricea de transformare de observare construită în matricea curentă este reținută în stivă prin cea de-a doua funcție `glPushMatrix()`. Matricea de instanțiere pentru prima planetă necesită introducerea mai întâi a rotației și apoi a



translației și ea se aplică acesteia. Matricea de instanțiere a satelitului se construiește prin compunerea matricei sale de instanțiere cu matricea de instanțiere a planetei respective. După instanțierea satelitului, matricea de instanțiere a acestuia nu mai este necesară, funcția `glPopMatrix()` o elimină din stivă și vârful stivei conține acum matricea de transformare de observare, care poate fi folosită pentru compunerea matricei pentru cea de-a doua planetă.



**Fig. 6.5** Două imagini ale sistemului solar, pentru două poziții diferite ale planetelor și satelitului.



În fig. 6.5 sunt prezentate două imagini captate din fereastra de afișare a programului pentru diferite valori ale pozițiilor planetelor și satelitului. Imaginea nefiind color, s-a considerat că reprezentarea wireframe pentru planete este mai sugestivă. Pentru fiecare obiect s-au desenat și axele de coordonate ale sistemului de referință local.

Este important de precizat faptul că, în marea majoritate a aplicațiilor grafice, fiecare imagine (cadru, *frame*) se generează independent de imaginile precedente. Pentru fiecare cadru se calculează poziția de observare și poziția obiectelor în scenă (dacă aceasta variază, așa cum este situația în exemplul sistemului solar) și imaginea se generează pornind de fiecare dată de la aceste informații. Trecerea de la o imagine la următoarea prin acumularea unor valori (de mișcare sau de poziție) nu aduce nici un beneficiu semnificativ ca viteză de execuție și poate produce erori de poziționare la execuție un timp îndelungat. Chiar modificarea cu o valoare extrem de mică a poziției primitivelor geometrice redată pe display necesită ștergerea imaginii precedente și construirea uneia noi prin reluarea întregii succesiuni a operațiilor grafice, deoarece este foarte complicat (practic imposibil) de a se calcula ce părți ale ecranului rămân neschimbate și ce părți se modifică și cum se modifică.

## 6.4 DECUPAREA OBIECTELOR ÎN OPENGL

În OpenGL, decuparea primitivelor geometrice la volumul de vizualizare este efectuată în mod automat, fără nici o intervenție din partea programatorului. Volumul de vizualizare se definește o dată cu definirea proiecției perspective (folosind una din funcțiile `glFrustum()` sau `gluPerspective()`) sau a proiecției paralele (folosind funcția `glOrtho()`). Dacă se urmărește diagrama de execuție din fig. 6.1, se poate observa că, după ce vârfurile au fost transformate prin înmulțire cu matricea curentă a stivei de modelare-vizualizare, se recompun primitivele geometrice, după care se aplică transformarea de normalizare și decuparea. În cursul transformării de modelare-vizualizare, vârfurile au fost tratate individual (transformat fiecare în parte), dar pentru decupare este necesar să fie recompuse în primitive, deoarece decuparea se efectuează prin parcurgere în ordine a laturilor poligoanelor.

Fiind o bibliotecă de funcții de nivel scăzut, care se ocupă de redarea unor primitive geometrice, OpenGL nu prevede nici o facilități pentru eliminarea obiectelor care sunt în exteriorul volumului de vizualizare (*culling*). Este sarcina programatorului să definească volumele de delimitare, să le transforme din sistemul local în sistemul de referință de observare și să le utilizeze pentru eliminarea obiectelor aflate complet în exteriorul volumului de vizualizare (obiecte invizibile).

Diferite sisteme de dezvoltare a aplicațiilor grafice (cum sunt sistemele Sense8 și Performer) oferă suport pentru definirea și utilizarea volumelor de delimitare și eliminare a obiectelor invizibile.



### 6.4.1 ELIMINAREA SUPRAFETELOR ASCUNSE

În OpenGL este implementat algoritmul Z-buffer de eliminare a suprafețelor ascunse. Numărul de biți/pixeli ai bufferului de adâncime se stabilește la definirea formatului pixelului. Implicit, se setează bufferul de adâncime cu 32 biți/pixel. Dacă se folosește utilitarul GLUT, atunci funcția de inițializare a modului de afișare trebuie să conțină opțiunea GLUT\_DEPTH:

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
```

La inițializarea bibliotecii OpenGL testul de adâncime este invalidat. Pentru validarea lui se apelează funcția `glEnable(GL_DEPTH_TEST)`, iar la începutul fiecărui cadru de imagine trebuie să fie șters și bufferul de adâncime, o dată cu ștergerea bufferului de imagine, ceea ce se obține prin introducerea opțiunii `GL_DEPTH_BUFFER_BIT` în apelul funcției `glClear()`:

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Valoarea cu care se “șterge” bufferul de adâncime este implicit valoarea maximă 1, iar testul care se efectuează implicit este de înscriere a noului pixel dacă are adâncimea mai mică decât valoarea din buffer (`GL_LESS`). În OpenGL mai există posibilitatea de a defini altă valoare de ștergere a bufferului de adâncime (prin funcția `glClearDepth(GLclampd depth)`), ca și stabilirea testului de comparație, prin funcția

```
void glDepthFunc(GLenum func);
```

unde `func` poate fi una din mai multe constante simbolice admise (`GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`).

### 6.4.2 SELECTIA SUPRAFETELOR ÎN FUNCȚIE DE ORIENTARE

OpenGL oferă suport direct pentru selecția primitivelor geometrice în funcție de orientarea lor. O primitivă geometrică este de categoria cu orientare directă (spre față - *frontface*) dacă sensul de parcurgere al acesteia (ordinea vârfurilor date prin funcțiile `glVertex#()` cuprinse în blocul `glBegin()`, `glEnd()`) este aceeași cu ordinea stabilită prin funcția `glFrontFace()` (în sensul acelor de ceas sau în sens invers acelor de ceas). Altfel, primitiva are orientare inversă (spre spate - *backface*). Prin funcția `glCullFace(GLenum type)` se selectează care tip de suprafețe trebuie eliminate: dacă argumentul `type` are valoare `GL_BACK` sunt eliminate primitivele cu orientare inversă; dacă are valoarea `GL_FRONT` sunt eliminate primitivele cu orientare directă. Testul de orientare și eliminarea primitivelor se execută numai dacă a fost validat acest test, prin apelul funcției `glEnable(GL_CULL_FACE)`.

În segmentul de cod care urmează, cele două poligoane sunt cu orientare directă (*front*), deoarece vârfurile lor sunt date în sensul invers acelor de ceas, așa cum este setarea implicită a sensului de parcurgere directă în OpenGL. Primul



poligon este desenat, deoarece este activă opțiunea de eliminare a primitivelor cu orientare inversă, iar al doilea nu este desenat, deoarece este activă opțiunea de eliminare a primitivelor cu orientare directă.

```
void Display() {
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glEnable(GL_CULL_FACE); // validează selecția
    glCullFace(GL_BACK); // se elimină primitive inverse
    glColor3f(0.5, 0.5, 0.5);
    glBegin(GL_POLYGON); // orientat direct, se desenează
        glVertex3d(3, 3, -10);
        glVertex3d(4, 3, -10);
        glVertex3d(4, 4, -10);
        glVertex3d(3, 4, -10);
    glEnd();
    glCullFace(GL_FRONT); // se elimină primitive directe
    glBegin(GL_POLYGON); // orientat direct, se elimină
        glVertex3d(1, 1, -10);
        glVertex3d(2, 1, -10);
        glVertex3d(2, 2, -10);
        glVertex3d(1, 2, -10);
    glEnd();
}
```

## 6.5 LISTE DE DISPLAY OPENGL

O listă de display este un grup de comenzi OpenGL care sunt memorate pentru a fi utilizate ulterior. Atunci când este apelată o listă de display, comenzile pe care le conține sunt executate în ordinea în care ele au fost memorate. Listele de display pot îmbunătăți performanțele de execuție atunci când se redesenează aceleași forme geometrice de mai multe ori.

Definirea unei liste de display începe cu apelul funcției `glNewList()` și se termină la apelul funcției `glEndList()`. Prototipul funcției `glNewList` este:

```
void glNewList(GLuint list, GLenum mode);
```

Parametrul `list` este un indice care identifică lista de display și este folosit la apelul acesteia. Parametrul `mode` poate lua una din constantele simbolice `GL_COMPILE` sau `GL_COMPILE_AND_EXECUTE`. Modul `GL_COMPILE` are ca efect compunerea listei de display. Modul `GL_COMPILE_AND_EXECUTE` are ca efect compunerea listei și execuția ei imediată. O listă de display cu indicele `list` este executată la apelul funcției:

```
void glCallList(GLuint list);
```

Ca exemplu de utilizare a unei liste de display se reiau comenzile de desenare a unui octogon din exemplul 6.2.



```

#include <math.h>
#include <gl\glut.h>
#define PI 3.141592
void DispList1(){
    glNewList(1, GL_COMPILE);
    int n = 8;
    double radius = 10;
    glColor3d(0,0,0);
    glBegin(GL_LINE_LOOP);
        for (int i=0; i<n; i++){
            double angle = 2*PI*i/n;
            glVertex3d(radius*cos(angle),
                radius*sin(angle), -40);
        }
    glEnd();
    glEndList();
}
void Init(){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    DispList1();
}
void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glCallList(1);
    glutSwapBuffers();
}

```

Funcțiile `Reshape()` și `main()` ale programului sunt cele prezentate în programele precedente. Funcția `DisplayList()` apelată la inițializare crează lista de display cu indicele 1 care conține calculul coordonatelor octogonului și OpenGL de desenare a acestuia. În funcția `Display()` este apelată funcția `glCallList(1)` care redesenează octogonul la fiecare apel.

Avantajul listelor de display este semnificativ atunci când, pentru generarea unei imagini, se folosesc funcții de calcul combinate cu comenzi OpenGL: calculele se execută o singură dată, la crearea listei, iar comenzile OpenGL rezultate se execută la fiecare apel al listei. O astfel de situație este prezentă și în exemplul de mai sus și apare frecvent în operațiile de iluminare, umbrire și texturare a obiectelor.

Dacă programul creează o singură listă de display, atunci se poate folosi valoarea 1 pentru indicele list din apelul funcției `glNewList()` așa cum s-a procedat în exemplul precedent. Dacă se creează mai multe liste de display, indicii acestora sunt valori întregi succesive dintre care primul indice este obținut prin apelul funcției `glGenLists(GLsizei range)`.

Dacă valoarea returnată de funcția `glGenLists()` este numărul  $k$ , atunci indicii listelor care se pot crea au valorile  $k, k+1, \dots, k+range-1$ . La primul apel al funcției `glGenLists()` se obține întotdeauna valoarea 1, deci în programele care creează o singură listă de display se poate folosi direct indicele 1 al listei.



## MODELE DE REFLEXIE ȘI ILUMINARE

Redarea obiectelor tridimensionale prin suprafețe colorate uniform creează imagini nerealiste și dificil de interpretat. De exemplu, dacă se redă imaginea unei sfere approximate prin poligoane colorând fețele vizibile cu aceeași culoare albă, se obține un disc alb (fig. 7.1). Se poate vedea că o sferă neluminată arată la fel ca un disc bidimensional. Acest aspect nerealist apare datorită faptului că percepția celei de-a treia dimensiuni este mult influențată de modul de redare a iluminării obiectelor. Imaginea aceleiași sferei în care se ține seama de iluminare este mult mai sugestivă.

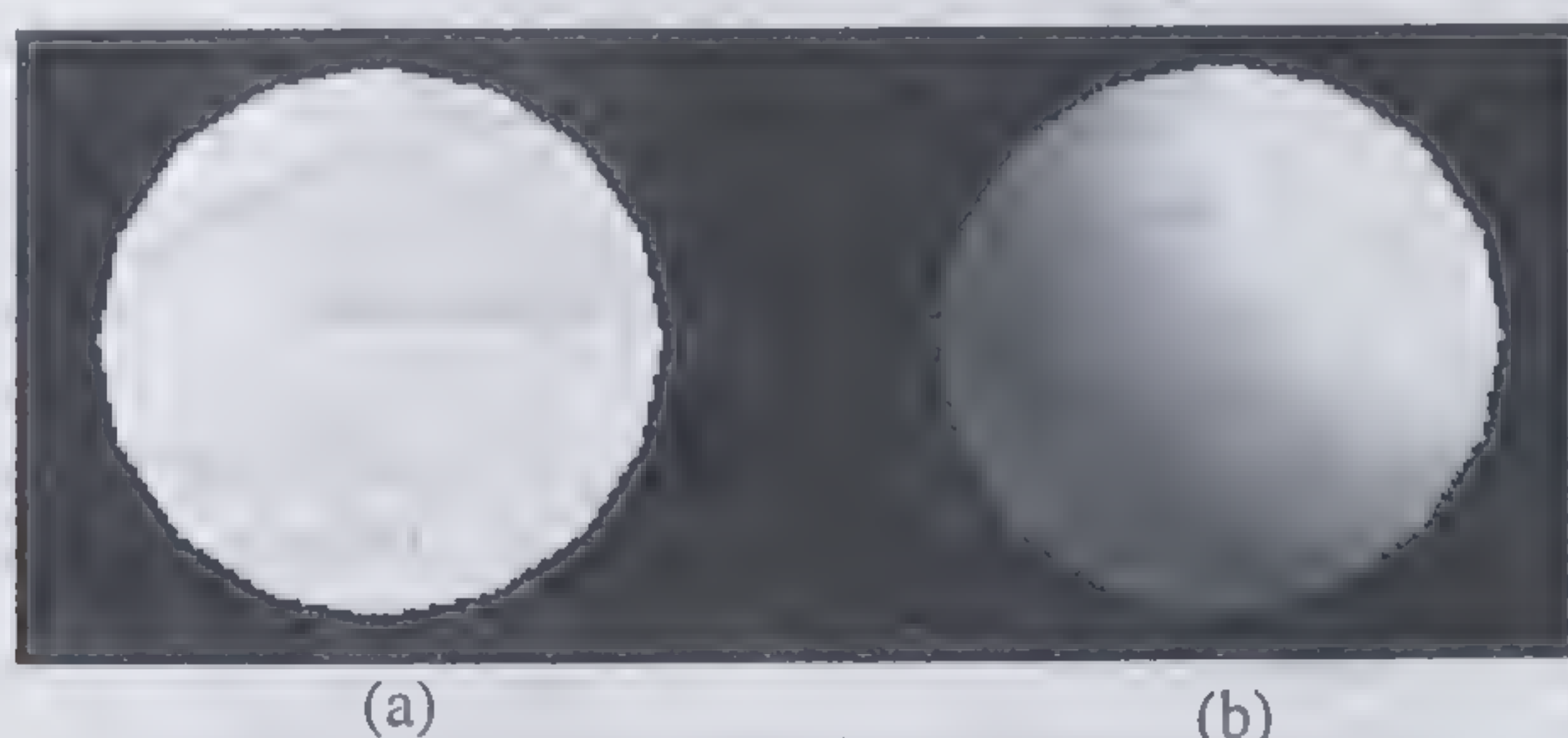


Fig. 7.1 Imaginea unei sfere: (a) fără iluminare; (b) cu iluminare.

În lumea reală, atunci când lumina provenită de la diferite surse de lumină cade asupra obiectelor opace, o parte este absorbită de obiect, iar o parte este reflectată. Ochiul percepe lumina reflectată de obiect, pentru a interpreta forma, culoarea și alte detalii ale obiectului. Pentru calculul iluminării în grafică pe calculator trebuie să fie definite sursele de lumină și interacțiunea dintre lumină și suprafețe.

Un *model de iluminare* definește natura luminii emise de o sursă de lumină, adică distribuția intensității luminii emise. Un *model de reflexie* descrie interacțiunea dintre lumină și o suprafață, în funcție de proprietățile suprafeței și natura sursei de lumină. Modelele de iluminare și de reflexie în grafică pe



calculator permit redarea acceptabilă din punct de vedere al percepției umane a obiectele tridimensionale proiectate în spațiul ecran bidimensional. Nivelul de acceptabilitate al redării depinde de natura aplicației. Cu cât este cerut un nivel de realism mai ridicat, cu atât este necesar un model de reflexie mai complex și cerințe de prelucrare mai mari.

Implementarea unui model de reflexie în procedeul de calculare a intensității culorii fiecărui pixel este cunoscută sub numele de *tehnica de umbrire*.

## 7.1 CONSIDERAȚII TEORETICE ASUPRA REFLEXIEI LUMINII

Lumina incidentă la o suprafață a unui obiect este distribuită în patru categorii: lumina reflectată, lumina adsorbită, lumina transmisă și lumina împrăștiată și emisă (fig. 7.2).

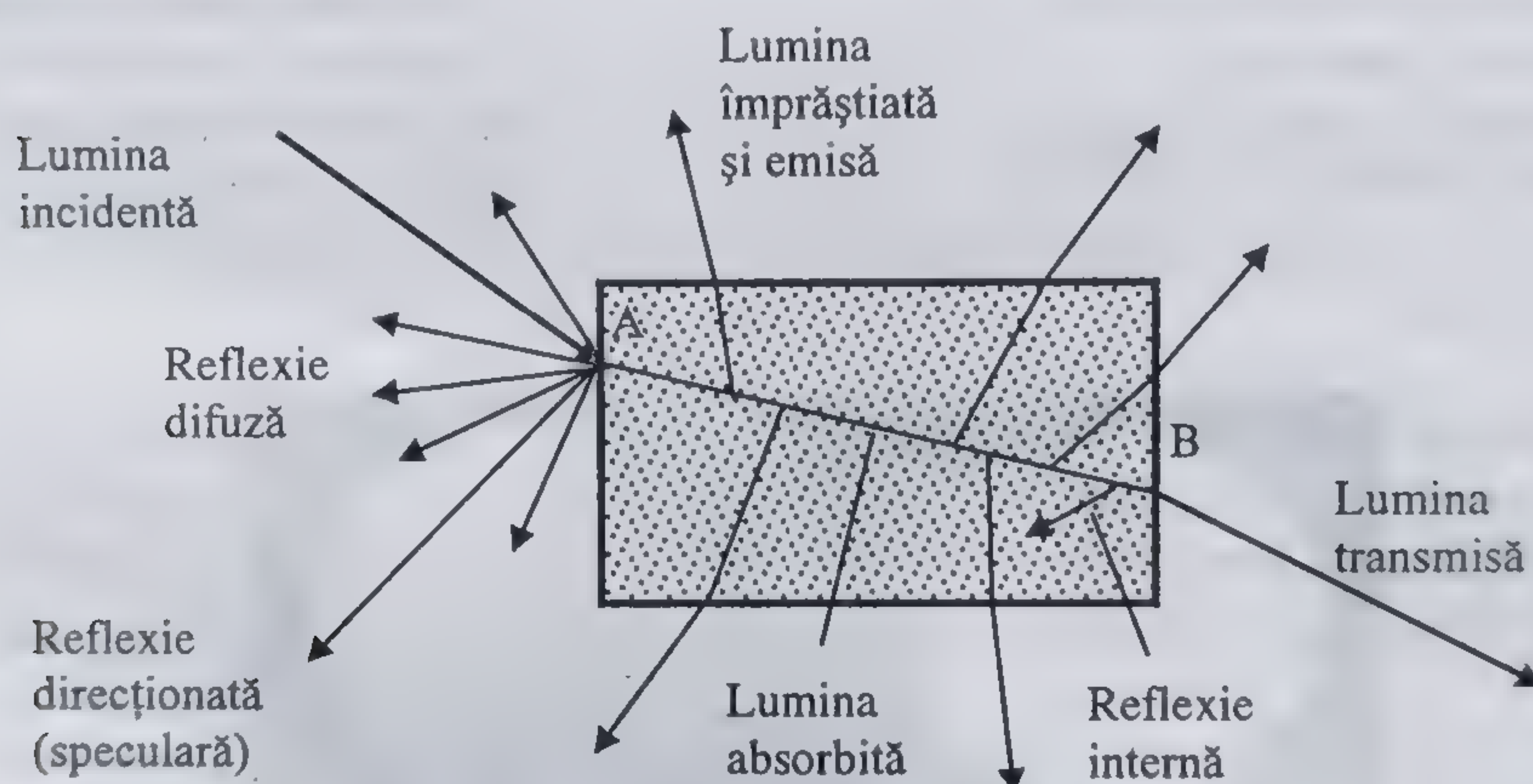


Fig. 7.2 Interacțiunea luminii cu un corp solid.

În sinteza de imagine se încearcă modelarea cât mai eficientă și completă a acestui mod de interacțiune a luminii cu obiectele. Intensitatea și lungimea de undă a luminii reflectate de o suprafață a unui obiect depinde de mai mulți factori: lungimea de undă a luminii incidente, unghiul de incidență, natura suprafeței și proprietățile ei electrice: permitivitatea, permeabilitatea și conductanța. Modelul exact al interacțiunii este extrem de complex și el poate fi aproximat prin intermediul *funcției de reflexie bidirecțională* (*bidirectional reflectivity function* – BDRF), care permite o apreciere cantitativă suficient de sugestivă pentru redarea iluminării obiectelor în grafica pe calculator.

Funcția de reflexie bidirecțională este relația dintre intensitatea luminii reflectate în direcția  $(\phi_v, \theta_v)$  și energia luminii primite din direcția  $(\phi_i, \theta_i)$  (fig. 7.3):



$$R_{bd}(\lambda, \phi_i, \theta_i, \phi_v, \theta_v) = \frac{I_v(\phi_i, \theta_i, \phi_v, \theta_v)}{E_i(\phi_i, \theta_i)} \quad (7.1)$$

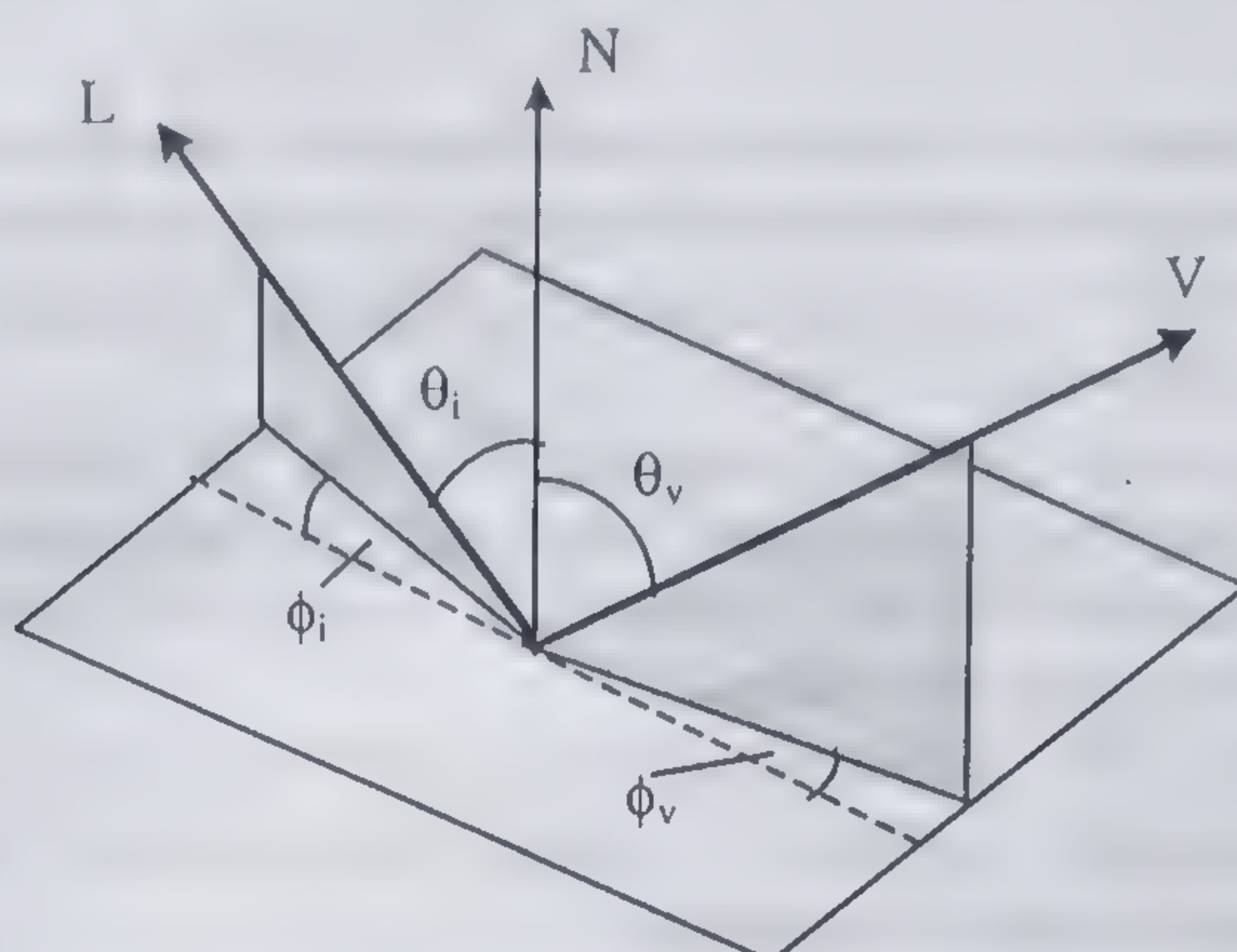


Fig. 7.3 Reflexia bidirecțională a luminii.

Funcția de reflexie bidirecțională depinde de lungimea de undă  $\lambda$ . Relația între energia luminii incidente corespunzătoare unui unghi solid  $\omega_i$  și intensitate  $I_i$  este:

$$E_i(\phi_i, \theta_i) = I_i(\phi_i, \theta_i) \cos \theta_i d\omega_i \quad (7.2)$$

În grafica pe calculator se consideră în mod simplificat că intensitatea reflectată este compusă din trei componente: componenta de reflexie direcționată (speculară), componenta de reflexie difuză direcționată și componenta de reflexie difuză ideală. Primele două componente se datorează reflexiei de prim ordin, iar componenta de reflexie difuză ideală se datorează reflexiilor multiple și a reflexiilor subsuprafețelor (fig. 7.4).

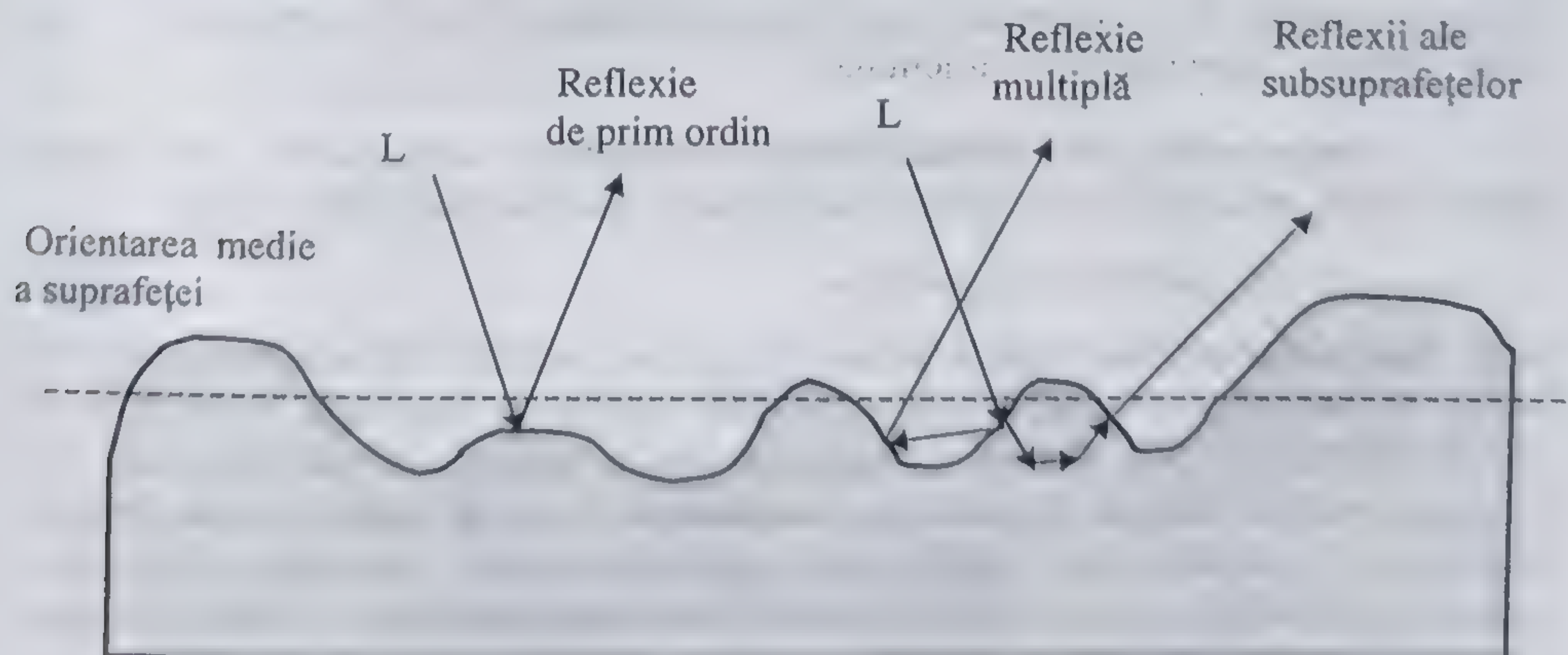


Fig. 7.4 Reflexii de prim ordin, reflexii multiple și reflexii ale subsuprafețelor



Valorile componentelor intensităților reflectate depind de rugozitatea suprafeței și de lungimea de undă a luminii. Dacă suprafața ar fi oglindă perfectă, atunci singura componentă de lumină reflectată ar fi componenta direcționată (speculară). Pentru suprafețe reale, cu un anumit grad de rugozitate, componenta speculară se definește ca intensitatea reflectată în direcția:  $\phi_v = \phi_i$ ,  $\theta_v = \theta_i$ , considerată față de orientarea medie a suprafeței. Intensitatea acestei componente depinde de rugozitatea suprafeței: cu cât suprafața este mai netedă, cu atât componenta speculară este mai mare.

Componenta de difuzie direcționată depinde de asemenea de reflexiile de prim ordin. Direcțiile de reflexie sunt împrăștiate în semisfera centrată în punctul de incidență, dar există o direcție preferințială, care este direcția reflexiei speculare pentru suprafețele cu rugozitate redusă.

Componenta de difuzie ideală reflectă lumina în toate direcțiile în interiorul unei semisfere centrate în punctul de incidență, datorită împrăștierii provocate de subsuprafețele componente.

Prin descompunerea funcției de reflexie bidirecțională în trei componente se poate dezvolta un model analitic bazat pe aspecte fizice, optice și geometrice, care să permită simularea iluminării și a reflexiei în grafica pe calculator. Modelul de reflexie Phong este cel mai cunoscut model în grafica pe calculator, care adoptă o reprezentare empirică și fenomenologică, prin formule de calcul simple de imitare a comportării teoretice a reflexiei luminii, descrise anterior [Phong75].

## 7.2 MODELUL DE REFLEXIE PHONG

Modelul de reflexie Phong imită eficient modul real de reflexie, până la un grad care produce o percepție destul de bună a obiectelor iluminate și, de aceea, are o largă utilizare în grafica pe calculator. Modelul de reflexie Phong consideră lumina reflectată de o suprafață ca fiind o combinație liniară a trei componente: reflexia difuză, speculară și ambientală.

**Componenta de reflexie difuză** corespunde componentei de reflexie difuză ideale descrise în subcapitolul precedent și este evaluată prin relația:

$$I_d = I_i k_d (\mathbf{L} \cdot \mathbf{N}) = I_i k_d \cos \theta \quad (7.3)$$

unde  $I_i$  este intensitatea luminii incidente,  $\mathbf{L}$  este vectorul unitate îndreptat către sursa de lumină și  $\mathbf{N}$  este vectorul unitate normal la suprafață în punctul de incidență (fig. 7.5(a)). Pentru o suprafață plană, unghiul  $\theta$  este constant, deci componenta de reflexie difuză este constantă.  $k_d$  este un coeficient de reflexie (reflectanță) empiric, care depinde de lungimea de undă a luminii și de natura suprafeței. Dat fiind că în reflexia difuză lumina este reflectată în toate direcțiile, această componentă nu depinde de poziția de observare.

Dacă există mai multe surse de lumină, atunci:



$$I_d = k_d \sum_n I_{i,n} (L_n \cdot N) \quad (7.4)$$

unde  $L_n$  este vectorul unitate în direcția celei de-a n-a surse de lumină.

Componenta speculară a reflexiei în modelul Phong depinde de unghiul  $\Omega$  între direcția de observare  $V$  și direcția de oglindire  $R$  (fig. 7.5(b)). Dacă se consideră  $R$  și  $V$  vectorii unitate ai acestor direcții, atunci:

$$I_s = I_i k_s \cos^n \Omega = I_i k_s (R \cdot V)^n \quad (7.5)$$

unde  $n$  este un indice care simulează rugozitatea suprafeței.

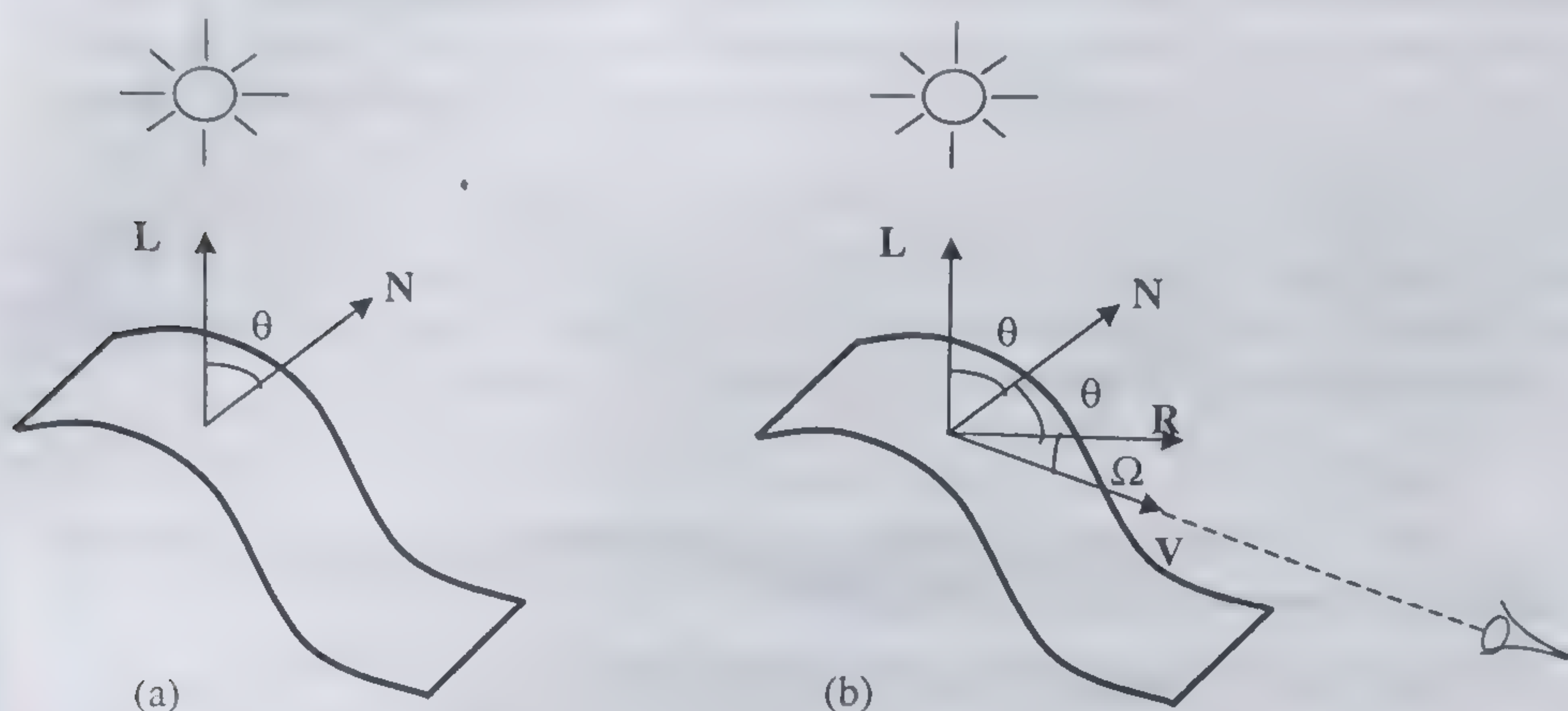


Fig. 7.5 (a) Reflexia difuză Phong. (b) Reflexia speculară Phong.

Pentru o suprafață oglindă perfectă,  $n$  tinde către infinit și lumina este reflectată numai în direcția de oglindire  $R$ , pentru care  $\cos^n \Omega = 1$ . Pentru valori finite ale indicelui  $n$ , se generează un lob de reflexie, a cărui grosime este o funcție de rugozitatea suprafeței (fig. 7.6).

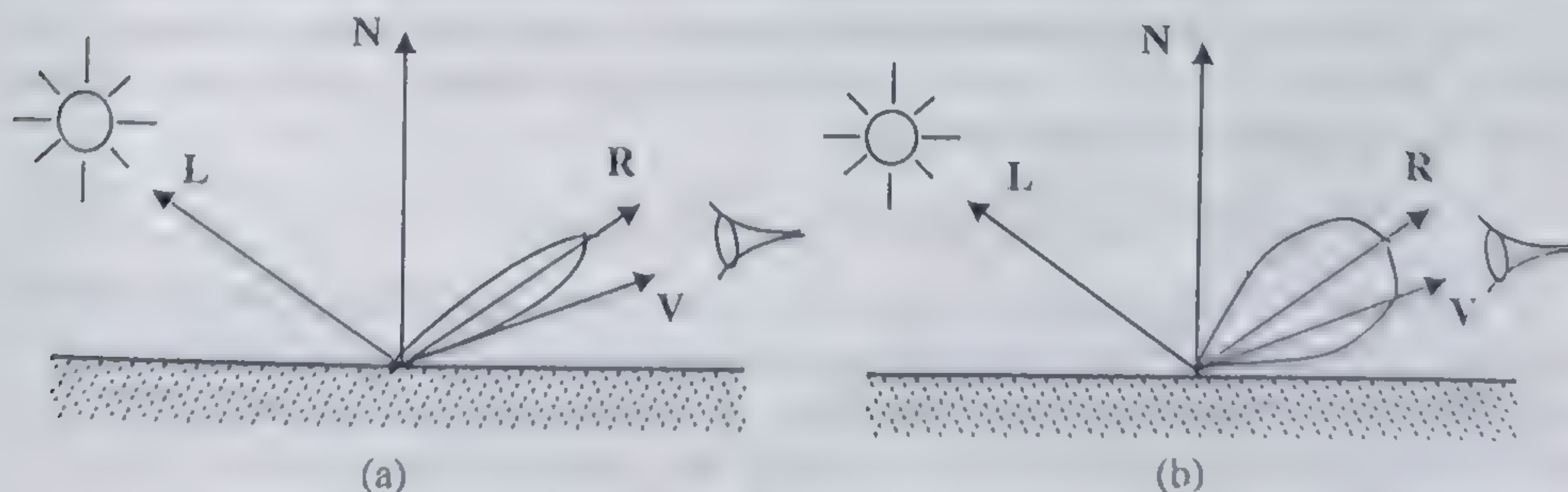


Fig. 7.6 (a) Indice  $n$  mare simulează reflexie speculară în lob îngust.  
(b) Indice  $n$  mic simulează reflexie speculară în lob larg.



Efectul reflexiei speculare în modelul Phong este acela de a produce o iluminare mai accentuată (*highlight*), care este reflexia sursei de lumină pe o arie a suprafeței, depinzând de valoarea lui  $n$  (Planșa 1). Culoarea luminii reflectate specular poate fi diferită de culoarea luminii reflectate difuz. În modelele cele mai simple de reflexie speculară, se presupune că această componentă are culoarea sursei de lumină. De exemplu, o suprafață de culoare verde iluminată de o sursă de lumină albă produce o componentă de reflexie de difuzie de culoare verde, dar pata luminoasă de reflexie speculară are culoarea albă.

**Componenta de lumină ambientală** se adaugă în modelul Phong pentru a simula iluminarea globală a scenei. Această componentă permite iluminarea suprafețelor care nu sunt vizibile din sursa de lumină, dar sunt vizibile din punctul de observare. Fără componenta ambientală, suprafețele care nu sunt îndreptate către o sursă de lumină ar fi redată complet negre, ceea ce ar crea un efect nerealist. Componenta ambientală aproximează iluminarea indirectă a suprafețelor, de exemplu datorită reflexiilor multiple față de pereții unei incinte. Aceste reflexii sunt foarte de complicat de simulat, astfel încât modelul Phong le ignoră și adaugă o componentă de lumină ambientală constantă dată de relația:

$$I_g = I_a k_a \quad (7.6)$$

Prin însumarea componentelor de lumină difuză, speculară și ambientală, se obține iluminarea totală a unei suprafețe:

$$I = I_a k_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n) \quad (7.7)$$

În modelul RGB de reprezentare a culorilor, relația (7.7) se descompune în trei relații, pentru fiecare componentă roșu, verde, albastru:

$$\begin{aligned} I_r &= I_a k_{ar} + I_i (k_{dr} (L \cdot N) + k_{sr} (R \cdot V)^n) \\ I_g &= I_a k_{ag} + I_i (k_{dg} (L \cdot N) + k_{sg} (R \cdot V)^n) \\ I_b &= I_a k_{ab} + I_i (k_{db} (L \cdot N) + k_{sb} (R \cdot V)^n) \end{aligned} \quad (7.8)$$

În mod obișnuit se consideră componenta speculară de culoare albă (deci  $k_{sr} = k_{sg} = k_{sb} = k_s$ ), iar culoarea suprafeței este controlată prin coeficienții de reflexie difuză ( $k_{dr}$ ,  $k_{dg}$ ,  $k_{db}$ ) și ambientală ( $k_{ar}$ ,  $k_{ag}$ ,  $k_{ab}$ ). Rezultă relația de calcul a culorii unei suprafețe în modelul Phong:

$$I(r, g, b) = I_a k_a(r, g, b) + I_i (k_d(r, g, b) (L \cdot N) + k_s (R \cdot V)^n) \quad (7.9)$$

Modelul Phong de reflexie este un model foarte simplificat al comportării obiectelor iluminate. În acest model, se consideră sursele de lumină punctiforme, ignorându-se distribuția surselor luminoase și simulându-se un termen global de iluminare ambientală. Cu toate aceste simplificări, modelul Phong produce imagini cu un grad de realism care este suficient pentru multe aplicații.

În Planșa 1 sunt reprezentate obiecte de culoare gri iluminate de surse de lumină de diferite intensități și culori. Se poate observa efectul reflexiei de difuzie, speculară și ambientale asupra aspectului obiectelor.



## 7.3 MODELE DE UMBRIRE

Aplicarea directă a relației (7.8) pentru calculul culorii fiecărui pixel necesită un timp de execuție extrem de ridicat, care nu este acceptabil în grafica interactivă. De aceea, în sinteza de imagine se folosesc anumite metode simplificate de calcul al culorii fiecărui pixel, numite tehnici (modele) de umbrire. Tehnicile de umbrire depind de modul de reprezentare a obiectelor. Pentru obiectele modelate prin rețea de poligoane, se folosesc mai multe modele de umbrire: umbrirea constantă (poligonală, *flat*), umbrirea Gouraud și umbrirea Phong.

### 7.3.1 MODELUL DE UMBRIRE CONSTANTĂ

În umbrirea constantă se admite calculul unei singure intensități a culorii pentru fiecare suprafață poligonală; acest mod de calcul este posibil dacă se admite ipoteza simplificatoare că atât sursele de lumină, cât și observatorul sunt la infinit. Umbrirea constantă produce discontinuități de culoare la frontiera dintre suprafețe. Calculul umbririi constante înseamnă aplicarea formulelor Phong (7.8) pentru fiecare suprafață vizibilă, o singură dată pentru fiecare poziție a observatorului.

### 7.3.2 MODELUL DE UMBRIRE GOURAUD

Pentru eliminarea discontinuităților de colorare care apar în umbrirea constantă, Gouraud a introdus o metodă de umbrire care-i poartă numele și care calculează intensitatea de culoare a pixelilor suprafețelor prin metode de interpolare, pornind de la intensitățile în vârfurile poligonului [Gour71]. Metoda incrementală de calcul al intensității de culoare a pixelilor este asemănătoare metodei de calcul al adâncimii pixelilor folosită în algoritmul Z-buffer. În general, se aplică combinat transformarea de rastru, eliminarea suprafețelor ascunse și umbrirea Gouraud, într-un algoritm de baleiere pe linii generalizat.

Umbrirea Gouraud este o tehnică de interpolare biliniară a intensității culorii, foarte simplă și economică, care atenuează discontinuitățile de la frontiera poligoanelor prin care este reprezentat un obiect tridimensional, fără să elimine complet aspectul poligonal al obiectelor.

Intensitatea de culoare se calculează în vârfurile poligoanelor prin aplicarea relațiilor (7.8) de calcul al reflexiei în modelul Phong. Normala într-un vârf al unui poligon se calculează ca medie a normalelor tuturor poligoanelor care sunt adiacente vârfului (fig. 7.7):

$$N_v = \sum_{i=1}^m \frac{N_i}{m} \quad (7.10)$$

Normalele în vârfurile poligoanelor sunt definite în sistemul de referință model și ele sunt transformate în sistemul de referință de observare prin aplicarea acelorași transformări care se aplică vârfurilor: transformarea de instanțiere și



transformarea de observare. În sistemul de referință de observare se calculează intensitățile în vârfurile poligoanelor și aceste valori sunt folosite pentru interpolare biliniară în algoritmul de conversie de baleiere pe linii a poligoanelor.

Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, și memorate ca parte a modelului obiectului în baza de date grafice. Atunci când se decupează fețele obiectelor, pot să apară vârfuri noi, care nu existau în modelul obiectului inițial. Pentru aceste vârfuri se calculează normalele în noile vârfuri prin interpolare între normalele laturilor intersectate de planul de decupare.

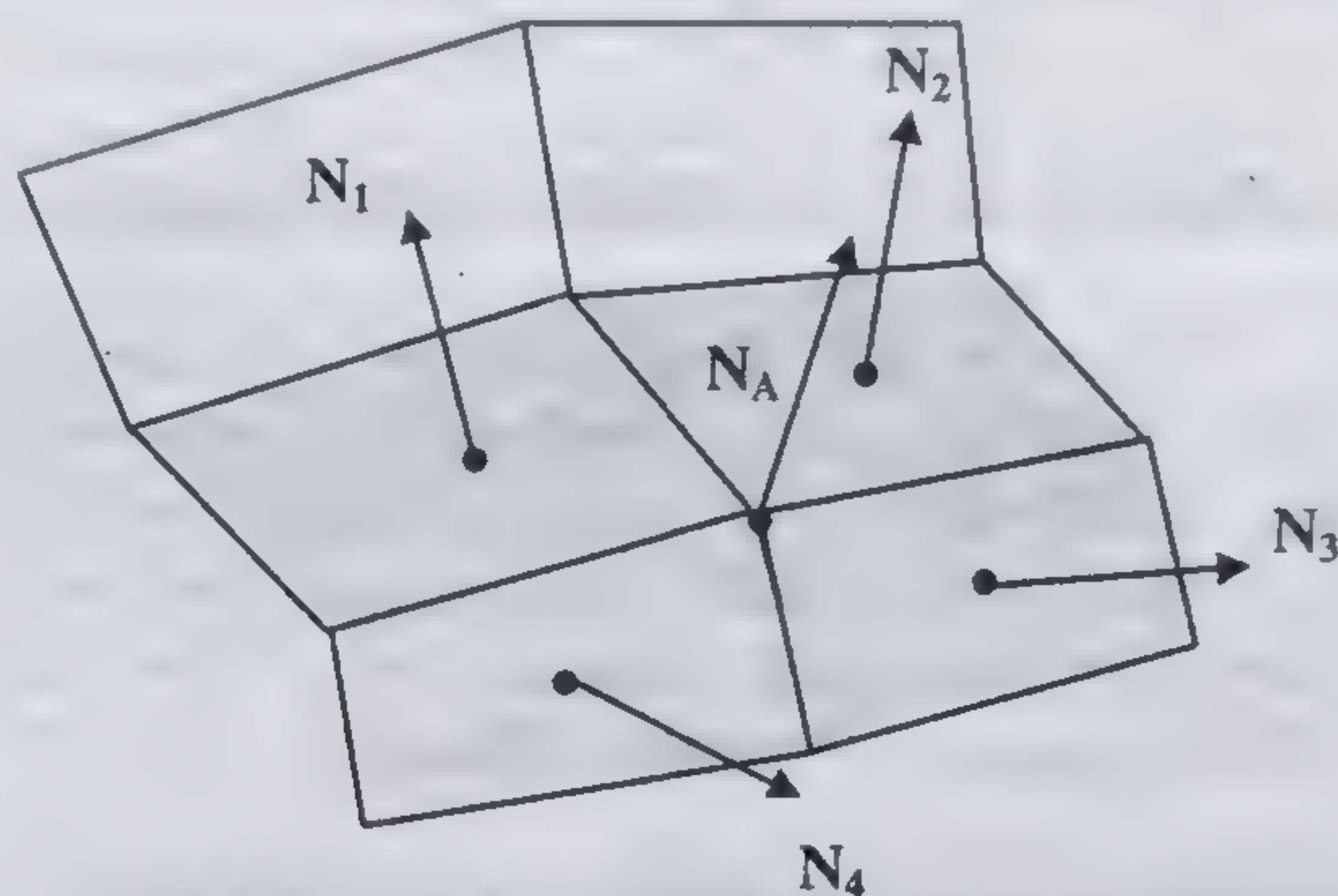


Fig. 7.7 Normala la vârful  $N_A$  este media normalelor  $N_1, N_2, N_3, N_4$  a poligoanelor adiacente în vârful respectiv.

Interpolarea biliniară a intensităților de culoare a pixelilor unui poligon se execută în cadrul algoritmului de baleiere pe linii pornind de la intensitățile de culoare ale vârfurilor poligonului, calculate prin relațiile modelului de reflexie Phong.

Se reia exemplul de baleiere pe linii din § 5.3 și 5.4.3. Poligonul ABCDE are intensitățile de culoare calculate în vârfuri:  $I_A, I_B, I_C, I_D$ , și, respectiv,  $I_E$ . Intensitățile la capetele liniilor de baleiere se calculează din intensitățile vârfurilor (fig. 7.8). Pentru laturile AB și BC, ecuațiile de interpolare sunt:

$$\begin{aligned} I &= m_{i1}y + n_{i1} \\ I &= m_{i2}y + n_{i2} \end{aligned} \quad (7.11)$$

$$\text{unde: } \begin{aligned} m_{i1} &= (I_A - I_B) / (y_A - y_B), \quad n_{i1} = y_B - m_{i1} I_B \\ m_{i2} &= (I_C - I_B) / (y_C - y_B), \quad n_{i2} = y_C - m_{i2} I_C \end{aligned} \quad (7.12)$$

Din motive de eficiență a calculelor, aceste ecuații se implementează incremental. Intensitățile se calculează în punctele de intersecție cu liniile de baleiere  $y_1, y_2, \dots, y_i, y_{i+1}, \dots$ . Fiind calculate intensitățile  $I_{i,1}$  și  $I_{i,2}$  corespunzătoare liniei de baleiere  $y = y_i$ , intensitățile  $I_{i+1,1}$  și  $I_{i+1,2}$ , corespunzătoare liniei de baleiere următoare,  $y_{i+1} = y_i + 1$  se obțin prin incrementare:

$$\begin{aligned} I_{i+1,1} &= I_{i,1} + m_{i1} \\ I_{i+1,2} &= I_{i,2} + m_{i2} \end{aligned} \quad (7.13)$$



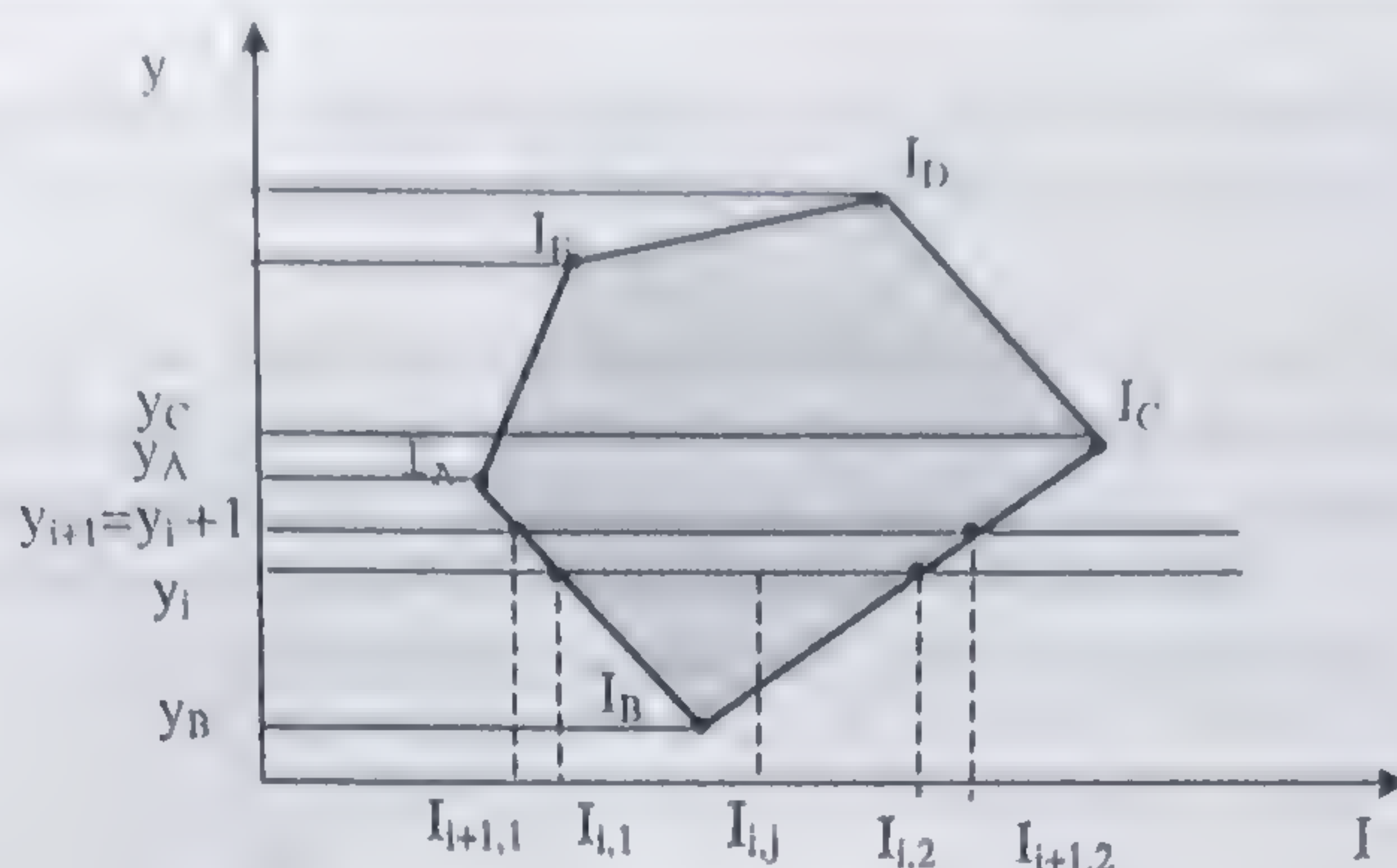


Fig. 7.8 Interpolarea intensității pe laturile poligonului și pe o linie de baleiere în umbrirea Gouraud.

Intensitatea  $I_{i,1}$  se atribuie pixelului a cărui adresă se obține din coordonatele punctului  $(x_{i,1}, y_i)$ , intensitatea  $I_{i,2}$  se atribuie pixelului a cărui adresă se obține din coordonatele punctului  $(x_{i,2}, y_i)$ , etc. Intensitățile de-a lungul liniei de baleiere se calculează din intensitățile la capetele acestuia. Pentru linia de baleiere  $y = y_i$ , ecuația de interpolare a intensității este:

$$I = m_{ix}x + n_{ix}$$

$$\text{unde: } m_{ix} = (I_{i,2} - I_{i,1}) / (x_{i,2} - x_{i,1}), \quad n_{ix} = x_{i,1} - m_{ix} I_{i,1} \quad (7.14)$$

Implementarea eficientă a acestei operații se face de asemenea incremental. Fiind calculată intensitatea  $I_{i,j}$  corespunzătoare punctului de coordonate  $(x_{i,j}, y_i)$ , intensitatea în punctul următor eșantionat pe linia de baleiere  $(x_{i,j+1}, y_i)$ , unde  $x_{i,j+1} = x_{i,j} + 1$  este:

$$I_{i,j+1} = I_{i,j} + m_{ix} \quad (7.15)$$

Pentru fiecare linie de baleiere se calculează intensitatea în puncte succesive prin incrementarea valorii precedente cu coeficientul dat de relația (7.14).

În fig. 7.9 este exemplificată diferența dintre umbrirea poligonală și umbrirea Gouraud. Două sfere reprezentate prin rețea de suprafețe plane (poligoane) sunt iluminate de la aceeași sursă punctiformă de lumină și au aceleași caracteristici de reflexie ale materialului.

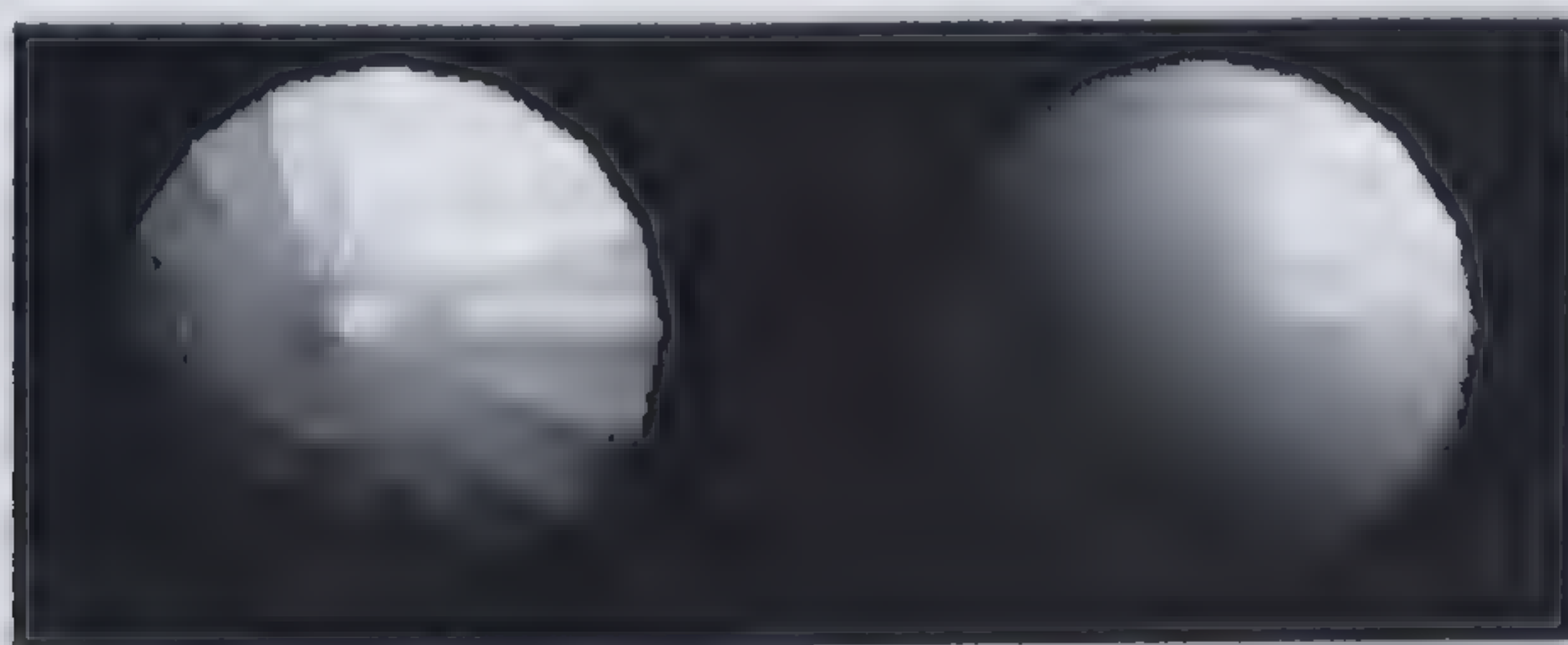


Fig. 7.9 Diferența dintre umbrirea poligonală și umbrirea Gouraud.



Diferența dintre cele două imagini ale sferei provine din faptul că pentru prima sferă se folosește umbrirea poligonală, iar pentru cea de-a doua sferă, umbrirea Gouraud.

Deși acceptabilă în multe aplicații, atât ca eficiență cât și ca realism de reprezentare, umbrirea Gouraud poate provoca unele anomalii provenind în primul rând din calculul normalei într-un vârf prin medierea normalelor fețelor adiacente. De exemplu, în fig. 7.10 se observă cum este posibil să se obțină normale în vârfuri identice (și deci intensități egale), din normale la suprafețe diferite.

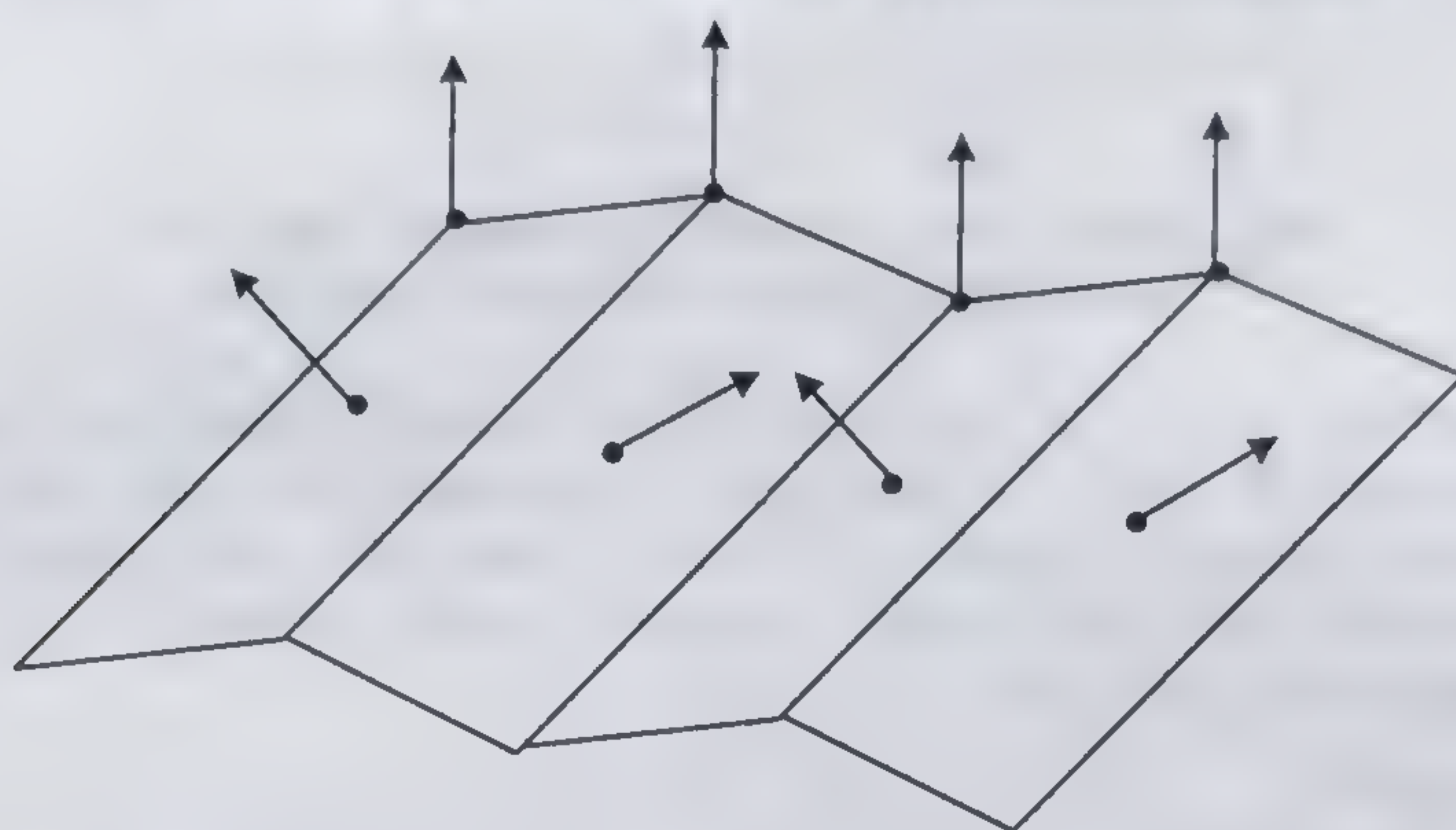


Fig. 7.10 O suprafață pliată regulat produce normale în vârfuri egale, deci colorare constantă a suprafețelor în umbrirea Gouraud

O altă deficiență a modelului Gouraud este aceea că nu se poate obține o pată de lumină (*highlight*) datorată reflexiei speculare în interiorul unui poligon, dacă vârfurile poligonului nu sunt cuprinse în această pată de lumină.

O parte din deficiențele tehnicii de umbrire Gouraud sunt eliminate în modelul de umbrire Phong.

### 7.3.3 MODELUL DE UMBRIRE PHONG

Modelul de umbrire Phong este, ca și modelul Gouraud, un model de calcul al intensității culorii prin interpolare biliniară, dar în acest model se interpolează normalele și se calculează exact intensitatea culorii (fig. 7.11).

Etapele de calcul a intensității culorii în modelul de umbrire Phong sunt:

- Calculul normalelor în vârfurile poligonului.
- Pentru fiecare linie de baleiere, se calculează prin interpolare vectorii normali ai intersecției liniei de baleiere cu laturile poligonului, folosind normalele în vârfuri ( $N_{i,1}$  și  $N_{i,2}$  în fig. 7.11).
- Normalele la capetele unei linii de baleiere sunt folosite pentru calculul prin interpolare a normalei fiecărui punct (cărui corespunde un pixel) de pe linia de baleiere ( $N_{ij}$ ).
- Normalele calculate prin interpolare sunt folosite pentru calculul intensității culorii în fiecare punct.



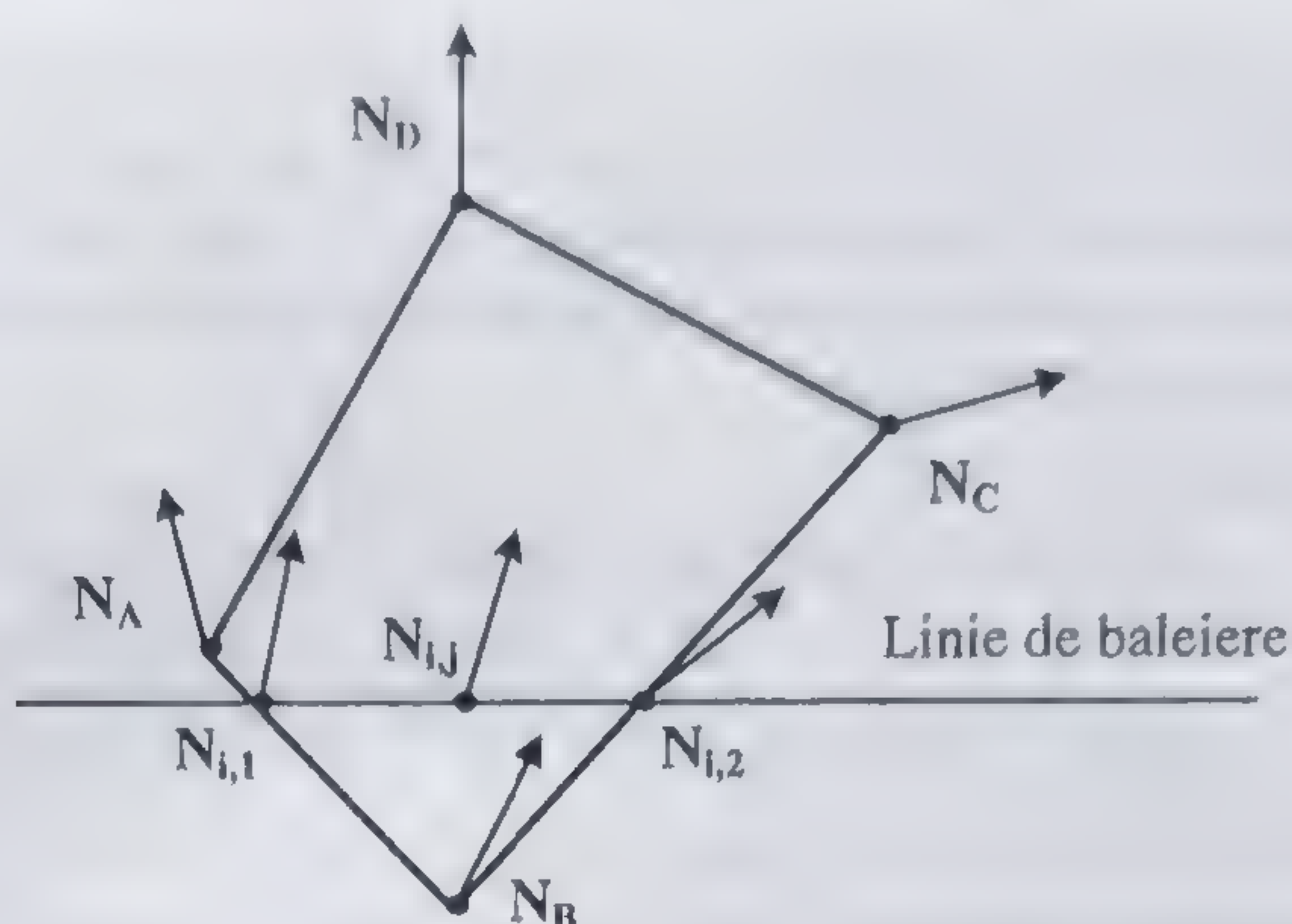


Fig. 7.11 Interpolarea normalelor în modelul de umbrire Phong.

Umbrirea Phong este mai costisitoare din punct de vedere al operațiilor efectuate deoarece, pentru fiecare pixel, se aplică relația de calcul (7.8), care conține produse scalare de vectori și înmulțiri. Dar, din punct de vedere al calității imaginii generate, umbrirea Phong este mai bună, dat fiindcă se calculează normale foarte apropiate de normala reală în fiecare punct al suprafeței. În tehnica de umbrire Phong se poate simula mult mai corect reflexia speculară (*highlight*).

În subcapitolul următor sunt prezentate modalitățile de programare a iluminării în aplicații grafice folosind biblioteca OpenGL.

## 7.4 GENERAREA FENOMENELOR NATURALE

Obiectele tridimensionale (clădiri, vehicule, etc) sunt ușor de modelat și de redat în grafica pe calculator și sunt intens folosite în scenele virtuale sau în sistemele de proiectare CAD. În aplicațiile de realitate virtuală în care scena este un spațiu geografic, realismul imaginii necesită însă și reprezentarea unor obiecte și fenomene naturale: teren, copaci, apă, foc, fum, nori, ceață, păclă, poluarea atmosferei, etc., fără de care imaginea se prezintă supărător de artificială. Sinteza obiectelor și a fenomenelor naturale este mult mai dificilă decât cea a obiectelor tridimensionale "normale", cu o formă bine precizată.

Pentru simularea efectelor atmosferice precum *ceață*, *păclă*, *poluare atmosferică*, modalitatea de reprezentare cea mai frecvent folosită este prin modificarea culorilor obiectelor din scenă pe baza distanței față de punctul de observare. Această operație este, de cele mai multe ori, implementată hardware în procesul de transformare de rastru a primitivelor geometrice. Pentru alte obiecte sau fenomene naturale există o varietate de reprezentări ad-hoc, care încearcă să obțină maximum de realism cu consum de resurse cât mai redus.

Ceața (ca și păcla sau poluarea atmosferică) este simulată prin combinarea culorii ceții cu culoarea fragmentelor suprafețelor, folosind un factor de combinare



care depinde de distanța acestora față de punctul de observare (coordonata  $z_s$  în sistemul de referință ecran 3D). Un fragment al unei suprafețe este porțiunea din suprafață vizibilă într-un pixel (care acoperă, deci, o parte sau întreg pixelul).

În simularea ceții se folosesc mai multe ecuații de calcul al factorului  $f$  de combinare a culorilor ( $0 \leq f \leq 1$ ):

$$f = e^{-(\text{dens} \cdot z_s)} \quad (7.16)$$

$$f = e^{-(\text{dens} \cdot z_s)^2} \quad (7.17)$$

$$f = \frac{\text{end} - z_s}{\text{end} - \text{start}} \quad (7.18)$$

În primele două ecuații, factorul de combinare variază exponențial, iar parametrul *dens* (un număr pozitiv) poate fi variat pentru obținerea unor efecte diferite de ceață. În cea de-a treia ecuație, factorul de combinare variază liniar cu distanța de la suprafață la punctul de observare, între limitele *start* (de unde “începe” ceața) pînă la limita *end* (unde “se termină” ceața).

Fiecare componentă  $C_r$  a culorii unui fragment ( $C$  poate fi R, G, B, A) se calculează prin combinarea culorii  $C_s$  a fragmentului cu culoarea  $C_f$  a ceții, prin ecuația:

$$C_r = fC_s + (1 - f)C_f \quad (7.19)$$

În combinarea liniară a culorilor pentru simularea ceții, la limita maximă *end*, nu se mai distinge nici un obiect, toate având culoarea unică a ceții. În general, limita *end* trebuie să fie corelată cu distanța planului de vizibilitate depărtat (*far*) din definirea sistemului de vizualizare (trunchiul de piramidă de vizualizare). Dat fiind că nu se generează imaginea nici unui obiect aflat la distanță mai mare decât distanța maximă de vizibilitate, limita *end* trebuie să fie mai mică sau egală cu distanța de vizibilitate maximă (*far*).

Introducerea fenomenului de ceață este necesară în special în scenele virtuale care modelează arii geografice în care se desfășoară simularea antrenamentelor de zbor sau de conducere a altor vehicule. Impresia creată de obiecte clar desenate chiar atunci când sunt la distanțe mari este nerealistă, iar simularea ceții îmbunătățește imaginea generată, făcând-o mai apropiată de imaginea reală, în care obiectele aflate la distanță mare se văd estompate. În acest mod, simularea ceții contribuie la percepția distanței obiectelor în scenă (*depth cueing*).

În funcție de parametrii de simulare a ceții, se pot obține diferite alte efecte. De exemplu, ceața de densitate redusă și care apare numai la distanțe foarte mari, capătă aspect de păclă; ceața de densitate redusă, distribuită uniform în scenă, poate fi considerată poluare atmosferică.

Un alt avantaj al simulării ceții îl reprezintă diminuarea efectului de aliasing și de aliasing al texturilor. Prin combinarea culorii suprafețelor cu culoarea ceții toate suprafețele sunt estompate și zgomotul de aliasing, chiar dacă este prezent, este mai puțin supărător. În cazul texturării, dat fiind că efectul de aliasing este mai pregnant atunci când suprafața este observată în perspectivă (deoarece



crește dimensiunea pre-imaginii pixelilor), estomparea culorilor cu creșterea distanței atenuază efectele de “moarăj” a unei suprafețe texturate privite în perspectivă. În exemplul 7.8 din subcapitolul următor este prezentat modul de obținere a efectului de ceață în aplicațiile grafice.

## 7.5 FUNCȚIILE OPENGL DE CALCUL AL ILUMINĂRII

Biblioteca OpenGL implementează modelul de reflexie Phong și modelele de umbrire poligonală și umbrire Gouraud. În mod implicit, sistemul de iluminare este inactivat și pentru desenarea primitivelor geometrice se folosește culoarea curentă, specificată prin funcția `glColor#()`. Pentru validarea iluminării obiectelor, se activează sistemul de iluminare OpenGL prin apelul funcției `glEnable (GL_LIGHTING)`.

Pentru calculul iluminării obiectelor trebuie să fie definite:

- sursele de lumină;
- materialul suprafețelor;
- modelul de umbrire a suprafețelor.

### 7.5.1 DEFINIREA SURSELOR DE LUMINĂ

În OpenGL se pot defini mai multe surse de lumină punctiforme. Numărul de lumini admise variază în funcție de implementare, dar cel puțin opt lumini sunt disponibile în orice bibliotecă OpenGL. Fiecare sursă de lumină poate fi validată prin apelul funcției `glEnable (GL_LIGHTi)`, unde  $i$  este indexul sursei de lumină.

O sursă de lumină se caracterizează prin intensitate și poziție în scenă. Intensitatea unei surse de lumină se specifică pentru fiecare componentă de iluminare (ambientală, de difuzie și speculară) printr-un vector în spațiul culorilor în modelul RGBA. Poziția unei surse se specifică printr-un vector în coordonate omogene corespunzătoare sistemului de referință universal.

Funcția de definire a unui parametru al unei surse de lumină este funcția `glLight#()` care are mai multe variante în funcție de tipul argumentelor. De exemplu:

```
void glLightfv(GLenum light, GLenum pname,  
               const GLfloat *params);  
void glLightiv(GLenum light, GLenum pname,  
               const GLint *params);
```

Argumentul `light` reprezintă indexul sursei de lumină și poate lua un nume simbolic de forma `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHTi`, unde  $0 \leq i < GL\_MAX\_LIGHTS$ . Numărul maxim de lumini depinde de implementarea bibliotecii.



Argumentul `pname` specifică un parametru al sursei de lumină. Sunt acceptate mai multe valori, dintre care unele se referă la intensitate de culoare iar altele la poziția sursei.

Valorile `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` ale argumentului `pname` permit definirea componentelor intensității culorii sursei de lumină. În această situație, argumentul `params` reprezintă un pointer la un vector de patru valori (de tip întreg sau virgula flotantă), care specifică intensitățile RGBA ale componente de iluminare ambientală, difuză și, respectiv, speculară. Valorile implicite ale intensităților sunt (0.0, 0.0, 0.0, 1.0) pentru componenta ambientală și pentru componentele difuză și speculară a oricărei lumini cu excepția luminii cu index 0, care are intensitatea difuză și speculară implicită (1.0, 1.0, 1.0, 1.0).

În acest model de definire a sursei de lumină, componenta ambientală (`GL_AMBIENT`) se referă la intensitatea RGBA pe care o sursă de lumină o adaugă iluminării globale a scenei. Componenta de iluminare difuză (`GL_DIFFUSE`) este cel mai apropiat mod de reprezentare a ceea ce se consideră culoarea sursei de lumină. Componenta de iluminare speculară (`GL_SPECULAR`) afectează culoarea zonei strălucitoare (*highlight*) a obiectelor luminate de sursa respectivă. Valoarea transparenței surselor de lumină (*alpha*) este ignorată dacă nu se validează calculul transparenței (*color blending*).

Valoarea `GL_POSITION` a argumentului `pname` permite definirea poziției sursei de lumină. În această situație, argumentul `params` este un pointer la un vector de patru numere (întregi sau virgula flotantă) care reprezintă poziția în coordonate omogene în sistemul de referință universal a sursei de lumină. Această poziție este transformată prin aplicarea valorii matricei de modelare-vizualizare din momentul apelului funcției `glLight#()`, astfel încât sursa de lumină va avea coordonatele transformate în sistemul de referință de observare, unde se calculează intensitatea în vârfurile primitivelor geometrice.

Dacă componenta `w` a poziției este 0, atunci lumina este tratată ca o lumină direcțională plasată la infinit, în direcția definită de componentele `x`, `y`, `z` și se utilizează această direcție pentru calculul componentelor de reflexie difuză și speculară.

Dacă  $w \neq 0$ , atunci sursa este o sursă pozițională și se folosește localizarea acesteia pentru calculul direcției de iluminare a suprafețelor. Poziția implicită a unei surse este (0, 0, 1, 0), deci sursa este plasată la infinit pe axa `z` și ea luminează obiectele în direcția  $-z$ . Secvența de instrucțiuni pentru definirea intensității și poziției sursei de lumină de indice 0 arată astfel:

```
GLfloat light_ambient[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```



În iluminarea reală, intensitatea luminii primite de un obiect scade odată cu creșterea distanței acestuia față de sursa de lumină. Dat fiind că sursele de lumină direcționale (care au  $w = 0$ ) se consideră plasate la infinit, nu are sens calculul atenuării cu distanța, astfel încât atenuarea este invalidată pentru astfel de surse. Pentru sursele poziționale, se folosește un factor de atenuare cu valoarea:

$$f = \frac{I}{k_c + k_l d + k_q d^2} \quad (7.20)$$

unde:  $d$  = distanța dintre sursa de lumină și vârful în care se calculează iluminarea

$k_c$  = GL\_CONSTANT\_ATTENUATION

$k_l$  = GL\_LINEAR\_ATTENUATION

$k_q$  = GL\_QUADRATIC\_ATTENUATION

Implicit, acești parametri au valorile:  $k_c = 1$ ,  $k_l = 0$ ,  $k_q = 0$ , dar ei pot fi setați prin apelul uneia din funcțiile `glLightf()` sau `glLighti()` astfel:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Dacă se folosesc mai multe surse de lumină, pentru fiecare dintre ele se definesc componentele intensității, poziția și caracteristicile de atenuare.

## 7.5.2 DEFINIREA PROPRIETĂȚILOR MATERIALELOR

Pentru calculul intensității culorii unei suprafețe, trebuie definiți coeficienții de reflexie pentru componentele de reflexie ambientală, difuză și speculară. Aceste proprietăți sunt considerate proprietăți de material al suprafeței și se specifică prin apelul uneia sau mai multora din cele patru variante ale funcției `glMaterial#()`:

```
void glMateriali(GLenum face, GLenum pname, GLint param);
void glMaterialf(GLenum face, GLenum pname,
                 GLfloat param);
void glMaterialiv(GLenum face,
                 GLenum pname, GLint *param);
void glMaterialfv(GLenum face, GLenum pname,
                 GLfloat *param);
```

În aceste funcții, argumentul `face` este numele feței și depinde de orientarea acesteia; poate lua ca valori constante simbolice `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`. Argumentul `pname` specifică proprietatea materialului, care se definește prin apelul funcției `glMaterial#()`. Acest argument poate lua una valorile date în tabelul 7.1.

Funcțiile `glMateriali()` și `glMaterialf()` se folosesc numai pentru setarea strălucirii (`GL_SHININESS`) la valoarea dată prin argumentul `param`.



Funcțiile `glMaterialiv()` și `glMaterialfv()` se folosesc pentru specificarea celorlalte proprietăți. Dacă argumentul `pname` este `GL_COLOR_INDEX` atunci `param` este un pointer la un vector de trei valori de tip întreg, care conține indicii culorilor ambientală, de difuzie și speculară ale materialului. În celelalte situații, `param` este un pointer la un vector de patru valori de tip întreg sau cu virgulă flotantă, care sunt componentele roșu, verde, albastru și alpha (transparență) ale componentei de reflexie specificate prin parametrul `pname` (ambientală, difuză, speculară, emisie). Se poate observa faptul că modelul OpenGL de reflexie permite adăugarea unei componente emiseive a luminii reflectate de o suprafață.

Tabelul 7.1

Parametrii de definire a materialelor

Nume parametru	Valoare implicită	Semnificație
<code>GL_AMBIENT</code>	(0.2, 0.2, 0.2, 1.0)	Reflectanța (coeficient de reflexie) ambientală
<code>GL_DIFFUSE</code>	(0.8, 0.8, 0.8, 1.0)	Reflectanța de difuzie
<code>GL_AMBIENT_AND_DIFFUSE</code>		Reflectanța ambientală și de difuzie
<code>GL_SPECULAR</code>	(0.0, 0.0, 0.0, 1.0)	Reflectanța speculară
<code>GL_EMISSION</code>	(0.0, 0.0, 0.0, 1.0)	Intensitatea luminii emise
<code>GL_SHININESS</code>	0.0	Exponentul de reflexie speculară
<code>GL_COLOR_INDEX</code>	(0, 1, 1)	Indicii culorilor ambientală, difuză și speculară ale materialului

**Reflexia difuză** joacă cel mai important rol în culoarea pe care o prezintă o suprafață. Ea reprezintă culoarea pe care o are suprafața luminată direct și depinde de componenta de difuzie a luminii incidente, provenită de la una sau mai multe surse de lumină, de coeficientul de reflexie de difuzie (reflectanța de difuzie) a materialului și de unghiul dintre direcția luminii și normala la suprafață. Poziția de observare nu influențează componenta de difuzie a luminii reflectate.

**Reflexia ambientală** afectează culoarea de ansamblu pe care o prezintă o suprafață și ea devine sesizabilă atunci când suprafața nu este luminată direct. Ca și reflexia difuză, această componentă nu depinde de poziția de observare. Cele două componente se specifică de cele mai multe ori cu aceeași culoare (așa cum sunt suprafețele reale) folosind parametrul `GL_AMBIENT_AND_DIFFUSE` în apelul funcției `glMaterial#()`.

**Reflexia speculară** produce iluminarea mai puternică (*highlight*) a unei zone a obiectului, în funcție de poziția de observare. OpenGL permite specificarea culorii produse de reflexia speculară (prin parametrul `GL_SPECULAR`) și a dimensiunii și strălucirii zonei prin parametrul `GL_SHININESS`.



**Componenta de emisie** a unei suprafețe se specifică prin parametrul `GL_EMISSION`. Acest efect este folosit pentru simularea lămpilor sau a altor surse de lumină din scenă.

**Proprietăților materialelor.** Funcția `glMaterial#()` definește proprietățile materialului curent, care se aplică tuturor vârfurilor introduse după aceasta prin funcțiile `glVertex#()` sau prin diferite funcții de modelare din biblioteca GLUT (de exemplu, `glutSolidSphere()`). Proprietățile materialului curent se mențin până la următorul apel al funcției `glMaterial#()`. Calculele de iluminare se pot executa diferit pentru fețele orientate direct (`GL_FRONT`) și cele orientate invers (`GL_BACK`). Un exemplu de definiție a unui material este dat prin următoarele instrucțiuni:

```
GLfloat mat_ambient[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat mat_shininess = 50.0;
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
```

**Modelul de umbrire** se definește prin apelul funcției `glShadeModel(GLenum mode)`, unde argumentul `mode` poate lua una din valorile `GL_FLAT`, pentru modelul de umbrire poligonală, sau `GL_SMOOTH`, pentru modelul de umbrire Gouraud. Valoarea implicită este `GL_SMOOTH`.

Dacă nu este validat sistemul de iluminare (prin apelul funcției `glEnable(GL_LIGHTING)`) atunci culoarea care se atribuie vârfurilor primitivelor geometrice este culoarea curentă, setată prin apelul unei funcții `glColor#()`.

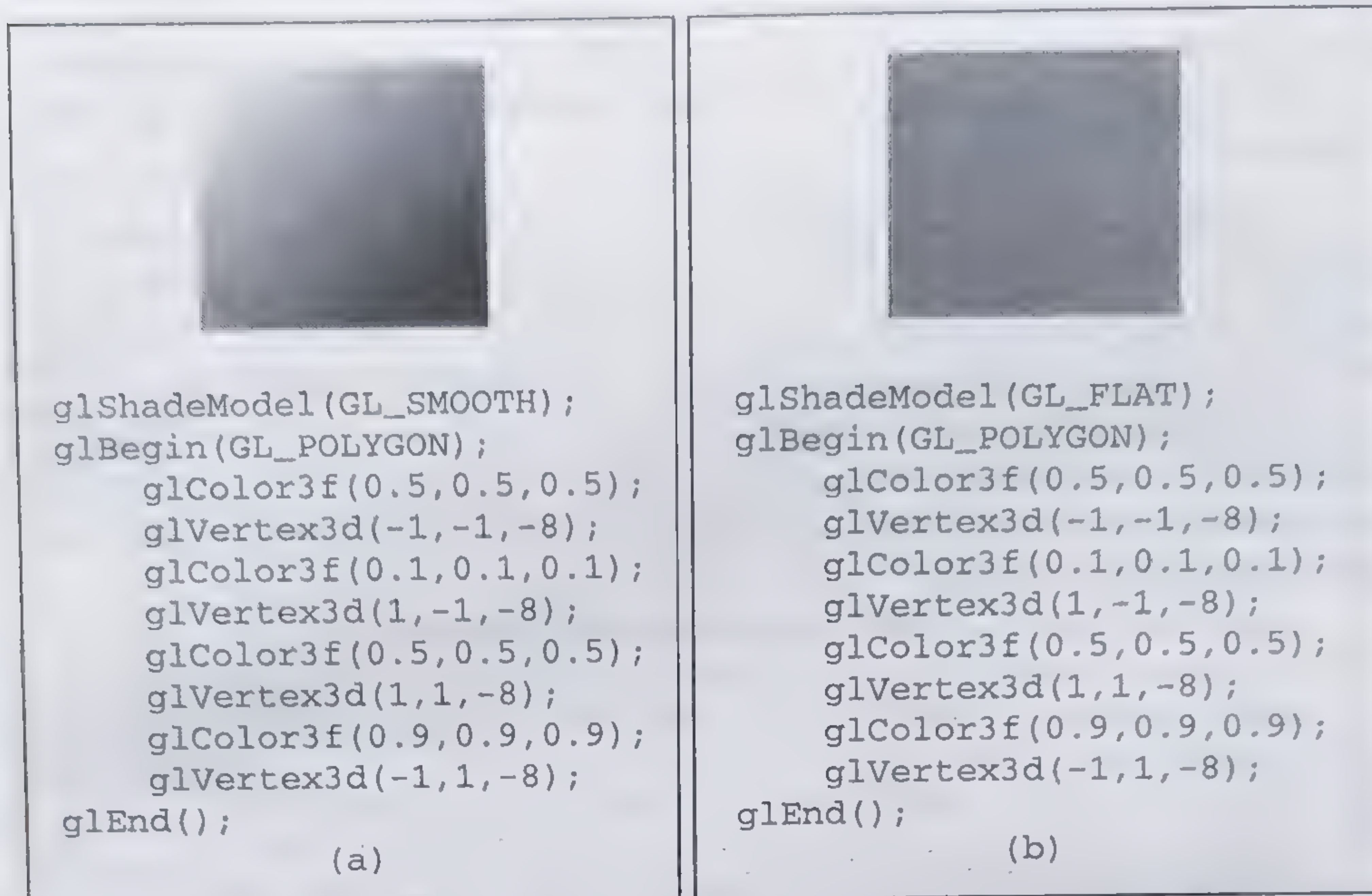
Dacă s-a definit modelul de umbrire poligonală (`GL_FLAT`), primitivele geometrice se generează de culoare constantă (culoarea curentă întâlnită la primul vârf introdus prin funcția `glVertex#()`). Dacă s-a definit modelul de umbrire Gouraud (`GL_SMOOTH`), atunci culorile definite în vârfuri se folosesc pentru calculul intensității culorii pixelilor primitivei prin interpolarea biliniară.

### ■ Exemplul 7.1

În fig 7.12 (a) este reprezentat un dreptunghi cu umbrire Gouraud (`GL_SMOOTH`), iar în fig 7.12 (b) este reprezentat același dreptunghi cu umbrire poligonală (`GL_FLAT`), toate celelalte funcții fiind aceleași.

Într-un mod asemănător, în Planșa 2 este reprezentat cubul culorilor RGB. Imaginile sunt obținute prin atribuirea culorilor roșu (001), verde (010), albastru (010), cian (011), magenta (101), galben (110), negru (000) și alb (111) vârfurilor cubului cu aceleași coordonate și prin validarea umbrii (`glShadeModel(GL_SMOOTH)`).





**Fig. 7.12** (a) Interpolarea biliniară a culorilor din vârfurile primitive.  
(b) Colorare constantă a primitivei.

Într-un mod asemănător, în Planșa 2 este reprezentat cubul culorilor RGB. Imaginile sunt obținute prin atribuirea culorilor roșu (001), verde (010), albastru (010), cian (011), magenta (101), galben (110), negru (000) și alb (111) vârfurilor cubului cu aceleași coordonate și prin validarea umbrii (`glShadeModel(GL_SMOOTH)`).

### ■ Exemplul 7.2

Pentru generarea unei imagini cu iluminare și umbrire (cum este cea din fig. 7.9), trebuie să se valideze sistemul de iluminare și să se definească cel puțin o sursă de lumină și parametrii materialului fețelor obiectelor. Programul care generează imaginea din fig. 7.9, scris folosind sistemul de dezvoltare GLUT este următorul:

```
#include <GL/glut.h>
void Init(){
    GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_ambient[] = {0.1, 0.1, 0.1, 1.0};
    GLfloat mat_diffuse[] = {0.9, 0.9, 0.9, 1.0};
    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess = 50.0 ;
    glClearColor(0.0,0.0,0.0,0.0);
```



```

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
    glShadeModel(GL_SMOOTH);          // implicit
}
void Display(void) {
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glPushMatrix();
        glTranslatef(-1.5, 0, -8);      //transf. modelare
        glShadeModel(GL_FLAT);
        glutSolidSphere(1, 16, 16);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(1.5, 0, -8);      // transf. modelare
        glShadeModel (GL_SMOOTH);
        glutSolidSphere(1, 16, 16);
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1, 40.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH );
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Lights");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```



Funcțiile callback `Display()` și `Reshape()` au aceleași semnificații descrise în programele precedente. Sursa de lumină direcțională (plasată la distanță infinită) are culoarea albă, iar materialul are componenta speculară albă, iar componentele difuză și ambientală au nuanțe de gri. Pentru redarea primei sfere s-a setat umbrire poligonală (`GL_FLAT`), iar pentru cea de-a doua sferă umbrire Gouraud (`GL_SMOOTH`). Imaginea din fig. 7.9 evidențiază diferența dintre umbrirea poligonală și umbrirea Gouraud, precum și iluminarea accentuată (pata luminoasă) datorată reflexiei speculare.

Dacă în programul de mai sus se înlocuiește componenta de difuzie a materialului cu o culoare roșie prin instrucțiunea:

```
GLfloat mat_diffuse[] = {0.9, 0.1, 0.2, 1.0};
```

se obține imaginea colorată din Planșa 4, în care este mai evidentă deosebirea dintre umbrirea poligonală și umbrirea Gouraud.

Aspectul realist al materialelor din care sunt construite obiectele se obține prin combinarea proprietăților de reflexie difuză, speculară și ambientală ale materialului. În Planșa 3 sunt reprezentate 24 de obiecte ceainic, care par a fi din diferite materiale reale.

### 7.5.3 NORMALELE ÎN VÂRFURILE PRIMITIVELOR GEOMETRICE

Pentru calculul reflexiei difuze și speculare este necesar să fie definite normalele în vârfurile primitivelor geometrice. Normala care se folosește pentru calculul reflexiei într-un vârf este normala curentă, care se setează prin apelul unei variante a funcției `glNormal#()` și se memorează într-o variabilă de stare a bibliotecii, identificată prin constanta simbolică `GL_CURRENT_NORMAL`. O parte din prototipurile funcțiilor `glNormal#()` definite în fișierul header `gl.h` sunt:

```
void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);
void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz);
void glNormal3i(GLint nx, GLint ny, GLint nz);
void glNormal3dv(const GLdouble *v);
void glNormal3fv(const GLfloat *v);
```

Normala se poate specifica prin componentele `nx`, `ny`, `nz`, sau printr-un pointer la un vector care conține cele trei componente, de tipul cerut prin numele funcției `glNormal#()`.

Normalele se poate defini în orice loc în program, în exteriorul sau în interiorul unui bloc `glBegin()-glEnd()`. Dacă normala se definește o singură dată pentru o primitivă geometrică, această valoare este folosită pentru calculul intensităților tuturor vârfurilor primitivei geometrice. În mod normal însă, pentru calculul corect al intensităților culorii în vârfuri, este necesar ca normala să fie definită pentru fiecare vârf al primitivei geometrice, deci funcțiile `glNormal#()` se apelează alternant cu funcțiile `glVertex#()` în interiorul blocului `glBegin()-glEnd()`.



În exemplul precedent nu a fost necesară definirea normalelor, deoarece funcția `glutSolidSphere()` le definește în mod automat, dar, pentru calculul iluminării unui obiect definit printr-o mulțime de primitive date prin vârfurile lor, este necesar să fie definite toate normalele în vârfuri.

Normalele se specifică în același sistem de referință ca și vârfurile primitivelor geometrice, deci în sistemul de referință de modelare. Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, și memorate ca parte a modelului obiectului în baza de date grafice. Asupra lor se aplică în mod automat transformarea de modelare-vizualizare folosind valoarea matricei din vârful stivei de modelare-vizualizare, existentă în momentul apelului funcției `glNormal#()`. În felul acesta, normalele sunt transformate în sistemul de referință de observare, unde se calculează intensitățile componentelor de reflexie în vârfurile primitivelor geometrice.

### ■ Exemplul 7.3

În acest exemplu se evidențiază necesitatea definirii normalelor pentru calculul iluminării. În program este validat sistemul de iluminare și este definită o sursă de lumină direcțională și materialul, cu componente de reflectanță difuză și ambientală. Cubul din fig. 7.13(a) a fost generat cu definirea normalelor în fiecare vârf al fiecărei suprafețe. În fig. 7.13 (b), unde singura diferență este absența normalelor în vârfuri; se observă absența umbririi obiectului.

Programul care produce aceste imagini este foarte asemănător cu programul din exemplul 7.2 pentru toate funcțiile, mai puțin funcția `Display()`, care arată astfel:

```
void Display(){
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable (GL_NORMALIZE);
    glPushMatrix();
    glTranslated(-2,-2,-8); // transf. de observare
    glLightfv(GL_LIGHT0,
              GL_POSITION, light_position);
    glBegin(GL_POLYGON);
        glNormal3d(-1,1, 1);
        glVertex3d(-1,1, 1);
        glNormal3d( 1,1, 1);
        glVertex3d( 1,1, 1);
        glNormal3d( 1,1,-1);
        glVertex3d( 1,1,-1);
        glNormal3d(-1,1,-1);
        glVertex3d(-1,1,-1);
    glEnd();
    glBegin(GL_POLYGON);
        glNormal3d(-1,-1,1);
        glVertex3d(-1,-1,1);
        glNormal3d( 1,-1,1);
        glVertex3d( 1,-1,1);
```



```

        glNormal3d( 1, 1, 1);
        glVertex3d( 1, 1, 1);
        glNormal3d(-1, 1, 1);
        glVertex3d(-1, 1, 1);
    glEnd();
    .....
glPopMatrix();
glutSwapBuffers();
)

```



(a)



(b)

**Fig. 7.13** (a) Calculul iluminării folosind normalele în vârfuri.  
(b) Imaginea obținută dacă nu sunt definite normalele în vârfuri.

În funcția `Display()` sunt introduse numai două din cele șase fețe ale cubului; celelate trei se definesc în mod similar. Funcția `glEnable(GL_NORMALIZE)` are ca efect scalarea la valoarea unitate a vectorilor normalelor introduse prin funcțiile `glNormal#()`.

Dacă se comentează toate funcțiile `glNormal3d()` din programul anterior, se obține imaginea din fig. 7.13(b), în care nu se calculează umbrirea fețelor cubului.

#### 7.5.4 CONTROLUL POZIȚIEI ȘI AL DIRECȚIEI SURSELOR DE LUMINĂ

OpenGL tratează poziția și direcția surselor de lumină la fel cum este tratată poziția primitivelor geometrice, adică sursele de lumină sunt transformate cu aceleași matrice de transformare ca și primitivele geometrice. Atunci când este apelată funcția `glLight#(GL_LIGHTi, GL_POSITION vect)` pentru specificarea poziției sau direcției unei surse de lumină, poziția sau direcția este transformată de matricea de modelare-vizualizare curentă și este memorată în coordonate în sistemul de referință de observare. Acest lucru înseamnă că se poate modifica poziția sau direcția unei surse de lumină prin modificarea corespunzătoare a matricei de modelare-vizualizare. Matricea de proiecție nu modifică poziția surselor de lumină. Sursele de lumină pot fi tratate în trei moduri:

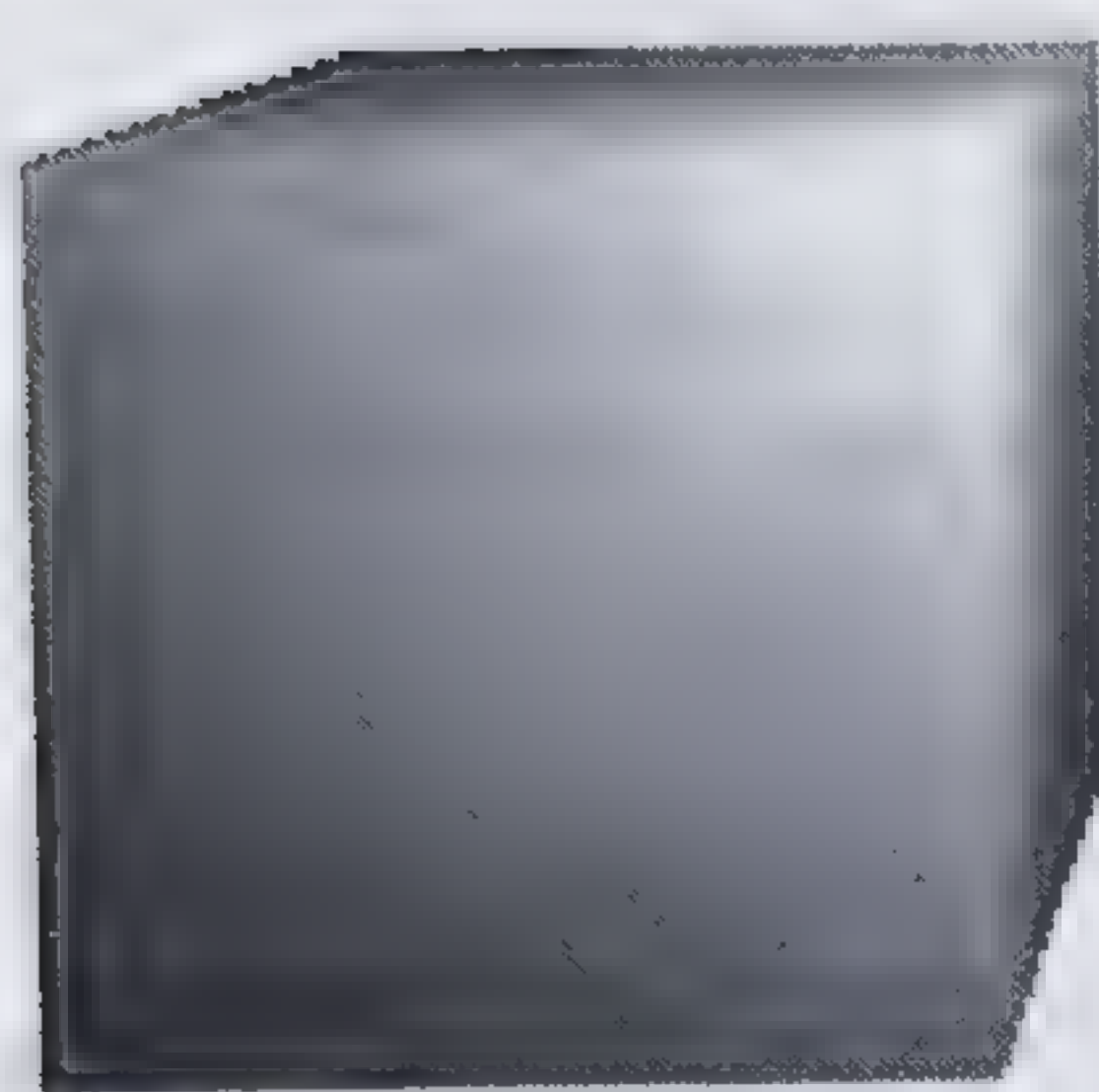
- surse cu poziție fixă;
- surse care se deplasează odată cu punctul de observare;
- surse care se deplasează în scenă.



```

        glNormal3d( 1, 1, 1);
        glVertex3d( 1, 1, 1);
        glNormal3d(-1, 1, 1);
        glVertex3d(-1, 1, 1);
    glEnd();
    .....
glPopMatrix();
glutSwapBuffers();
}

```



(a)



(b)

**Fig. 7.13** (a) Calculul iluminării folosind normalele în vârfuri.  
(b) Imaginea obținută dacă nu sunt definite normalele în vârfuri.

În funcția `Display()` sunt introduse numai două din cele șase fețe ale cubului; celelate trei se definesc în mod similar. Funcția `glEnable(GL_NORMALIZE)` are ca efect scalarea la valoarea unitate a vectorilor normalelor introduse prin funcțiile `glNormal#()`.

Dacă se comentează toate funcțiile `glNormal3d()` din programul anterior, se obține imaginea din fig. 7.13(b), în care nu se calculează umbrirea fețelor cubului.

#### 7.5.4 CONTROLUL POZIȚIEI ȘI AL DIRECȚIEI SURSELOR DE LUMINĂ

OpenGL tratează poziția și direcția surselor de lumină la fel cum este tratată poziția primitivelor geometrice, adică sursele de lumină sunt transformate cu aceleași matrice de transformare ca și primitivele geometrice. Atunci când este apelată funcția `glLight#(GL_LIGHTi, GL_POSITION vect)` pentru specificarea poziției sau direcției unei surse de lumină, poziția sau direcția este transformată de matricea de modelare-vizualizare curentă și este memorată în coordonate în sistemul de referință de observare. Acest lucru înseamnă că se poate modifica poziția sau direcția unei surse de lumină prin modificarea corespunzătoare a matricei de modelare-vizualizare. Matricea de proiecție nu modifică poziția surselor de lumină. Sursele de lumină pot fi tratate în trei moduri:

- surse cu poziție fixă;
- surse care se deplasează odată cu punctul de observare;
- surse care se deplasează în scenă.



**Surse de lumină cu poziție fixă.** Funcția `glLight#()` cu parametrul `GL_POSITION` definește coordonatele omogene ale poziției sursei. Dacă această funcție este apelată în momentul în care matricea curentă din stiva de modelare-vizualizare conține matricea de observare, atunci poziția sursei este transformată în coordonate de observare. Poziția în spațiu a sursei rămâne fixă, iar poziția calculată și utilizată în calcule este poziția în coordonate de observare. În exemplul 7.3 sursa de lumină este menținută în poziție fixă în scenă prin setarea poziției după introducerea transformării de observare:

```
glPushMatrix();
    glTranslated(-2,-2,-8); // transf. de observare
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    // Desenare obiect
glPopMatrix();
```

#### ■ Exemplul 7.4

Se reia programul de desenare a două sfere luminate din exemplul 7.2. Culoarea de ștergere este modificată (culoare gri deschis (0.8, 0.8, 0.8)) pentru o mai bună evidențiere a umbririi obiectelor. Transformările de instanțiere se aplică fiecărui obiect în parte. Transformarea de observare se aplică tuturor obiectelor și direcției sursei de lumină. În continuare este prezentată numai funcția `Display()` pentru evidențierea poziției fixe a sursei de lumină, în condițiile modificării punctului de observare ( $x_v = 0$ ,  $y_v = 0$ ,  $z_v = 8$  și unghiul  $\chi$  variabil).

```
void Display(){
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
        glTranslatef(0,0,-8);
        glRotated(gamma,0,1,0); // transf. observare
        glLightfv(GL_LIGHT0,
            GL_POSITION, light_position);
        glPushMatrix();
            glTranslatef(1.5,0,0); // transf. modelare
            glutSolidSphere(1,16,16);
        glPopMatrix();
        glPushMatrix();
            glTranslatef(-1.5,0,0); // transf. modelare
            glutSolidSphere(1,16,16);
        glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();
}
```

Prin mișcarea de rotație a punctului de observare, sferele sunt văzute iluminate în diferite situații: pentru  $\chi = 45^\circ$  se poate observa partea cea mai luminată a sferelor; pentru  $\chi = 225^\circ$  se observă partea cea mai puțin luminată a sferelor (fig. 7.14).



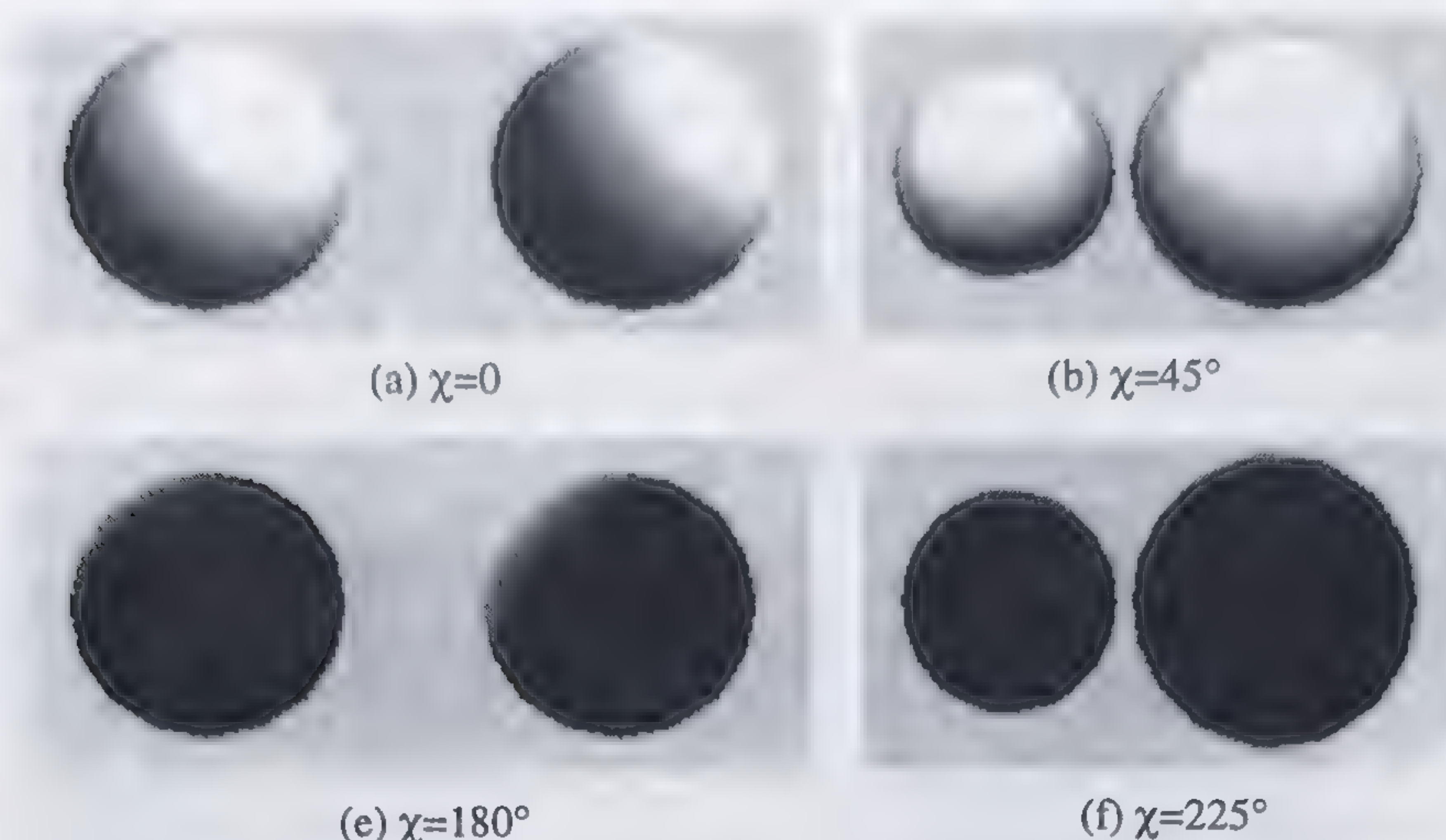


Fig. 7.14 Sursă de lumină fixă în direcția (1.0, 1.0, 1.0, 0.0).

**Surse de lumină care se deplasează odată cu punctul de observare.** Pentru a crea surse de lumină care se mișcă odată cu punctul de observare, trebuie ca poziția sursei să fie setată înainte de transformarea de observare. Poziția sau direcția sursei date prin funcția `glLight#()` cu parametrul `GL_POSITION` este transformată cu matricea curentă din stiva de modelare-vizualizare, care conține matricea identitate, deci localizarea sursei rămâne constantă față de punctul de observare. Acest mod de control al sursei este folosit în simulatoare de antrenament, pentru a simula lumina farurilor vehiculului. Dacă poziția sursei este (0,0,0,1), atunci sursa este plasată chiar în punctul de observare.

### ■ Exemplul 7.5

Se reia programul de generare a imaginii obiectelor luminate din exemplul precedent, dar sursa de lumină se menține cu o direcție fixă în sistemul de referință de observare, prin setarea direcției sursei înainte de transformarea de observare. În continuare sunt prezentate funcțiile callback `Display()` și `Keyboard()` ale programului, celelalte funcții fiind nemodificate (cu excepția culorii de ștergere care este gri închis) și a înregistrării funcției callback `Keyboard()`.

```
static int gamma = 0;
void Keyboard(unsigned char key, int x, int y){
    switch (key){
        case 'G':
            gamma = (gamma+30)%360;
            glutPostRedisplay();
            break;
        case 'g':
            gamma = (gamma-30)%360;
```



```

        glutPostRedisplay(); break;
    }
}

void Display() {
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glPushMatrix();
        glTranslatef(0,0,-8);
        glRotated(gamma,0,1,0); // transf. observare
        glPushMatrix();
            glTranslatef(1.5,0,0); // transf. modelare
            glutSolidTeapot(1);
        glPopMatrix();
        glPushMatrix();
            glTranslatef(-1.5,0,0); // transf. modelare
            glutSolidTeapot(1);
        glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();
}

```

Sursa de lumină direcțională are direcția (1.0, 1.0, 1.0, 1.0) care, transformată cu matricea identitate, păstrează aceeași valoare în sistemul de referință de observare. De aceea, indiferent de poziția și direcția de observare, obiectele sunt iluminate dintr-o direcție fixă față de direcția de observare.

În fig. 7.15 sunt reprezentate mai multe imagini, pentru direcții de observare diferite. Unghiul  $\chi$  (gamma în program) de rotație față de axa y se modifică prin comenzi de la tastatură, prelucrate de funcția callback `Keyboard()`.

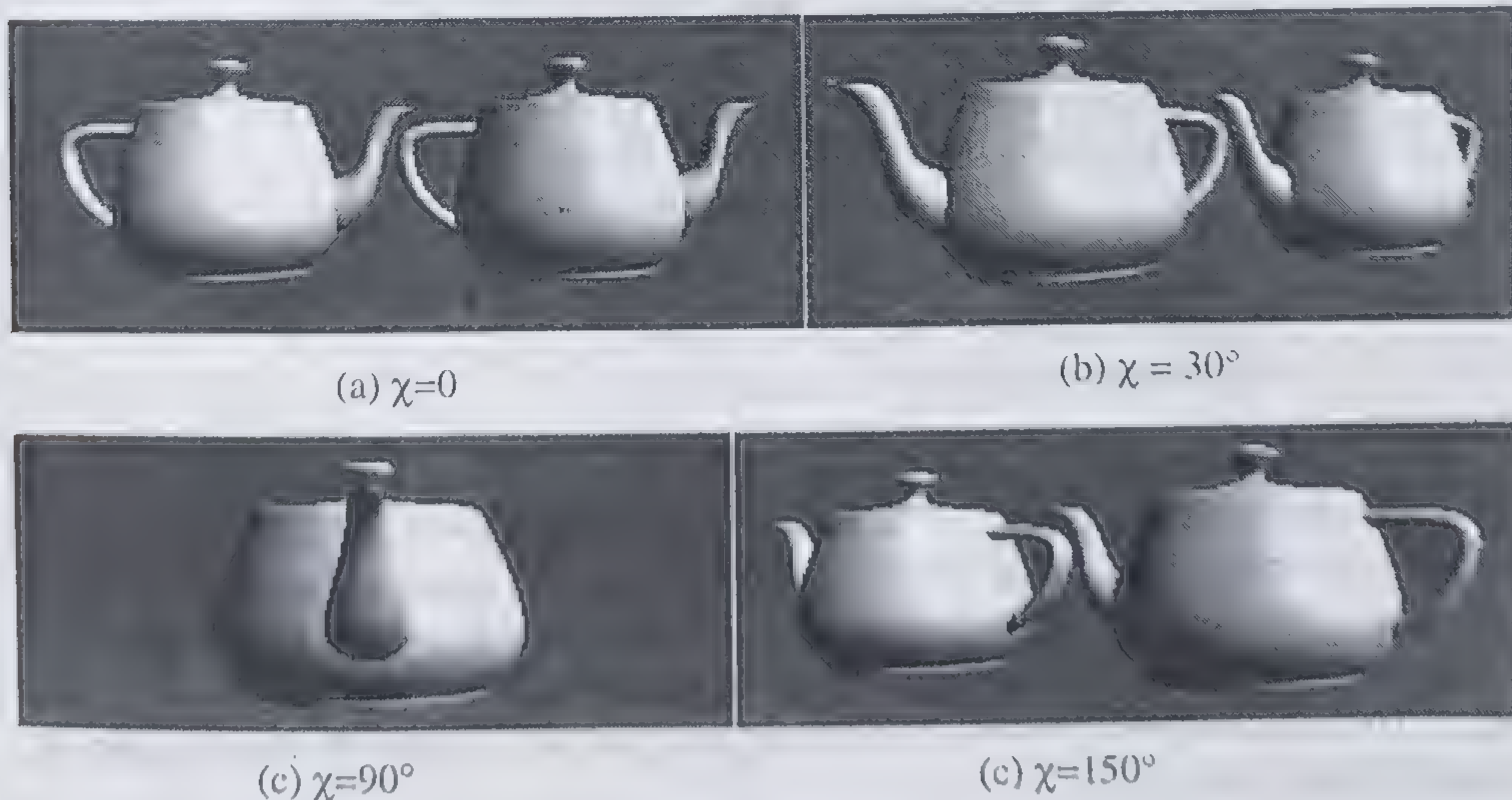


Fig. 7.15 Sursă de lumină care se deplasează odată cu punctul de observare.



Surse de lumină care se deplasează în scenă. Se pot defini surse de lumină care se mișcă independent în scena virtuală, prin aplicarea unei transformări de instanțiere (care localizează sursa de lumină în sistemul de referință universal), urmată de transformarea de observare, prin care se calculează poziția sau direcția sursei în sistemul de referință de observare. La fel ca și în cazul instanțierii obiectelor în scenă, se folosește matricea curentă din stiva de modelare-vizualizare.

### ■ Exemplul 7.6

Se consideră un punct și o direcție de observare fixă, iar obiectele din scenă sunt luminate de o sursă de lumină direcțională, care își modifică direcția printr-o comandă de la mouse. Programul care implementează această funcționare este similar celorlalte programe de iluminare prezentate în exemplele precedente, cu excepția funcției `Display()` și a funcției callback de prelucrare a evenimentelor de la mouse, care trebuie înregistrată în GLUT.

```
#include <GL/glut.h>
static int teta = 0;
void Mouse(int button, int state, int x, int y){
    switch(button){
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN){
                teta = (teta+30)%360;
                glutPostRedisplay();
            }
    }
}
void Display(){
    GLfloat light_position[] = {0, 0, 2, 1.0};
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(0,0,-8);    // transf. observare
    glPushMatrix();
    glRotated(teta,1,0,0); // transf. inst. sursa
    glLightfv(GL_LIGHT0, GL_POSITION,
              light_position);
    glPopMatrix();
    glutSolidTorus(0.3, 0.8, 8, 16);
    glPopMatrix();
    glutSwapBuffers();
}
```

Inițial variabila `teta = 0` și sursa este plasată pe axa `z` la distanța egală cu 2 față de centrul sistemului de referință universal. La fiecare apăsare a butonului stânga al mousului, variabila `teta` este incrementată (modulo 360) cu 30, și poziția sursei este rotită cu unghiul spin în raport cu axa `x`. Imaginile succesive (fig. 7.16) indică modul în care este iluminat un tor de o sursă de lumină care se rotește în jurul axei `x`.



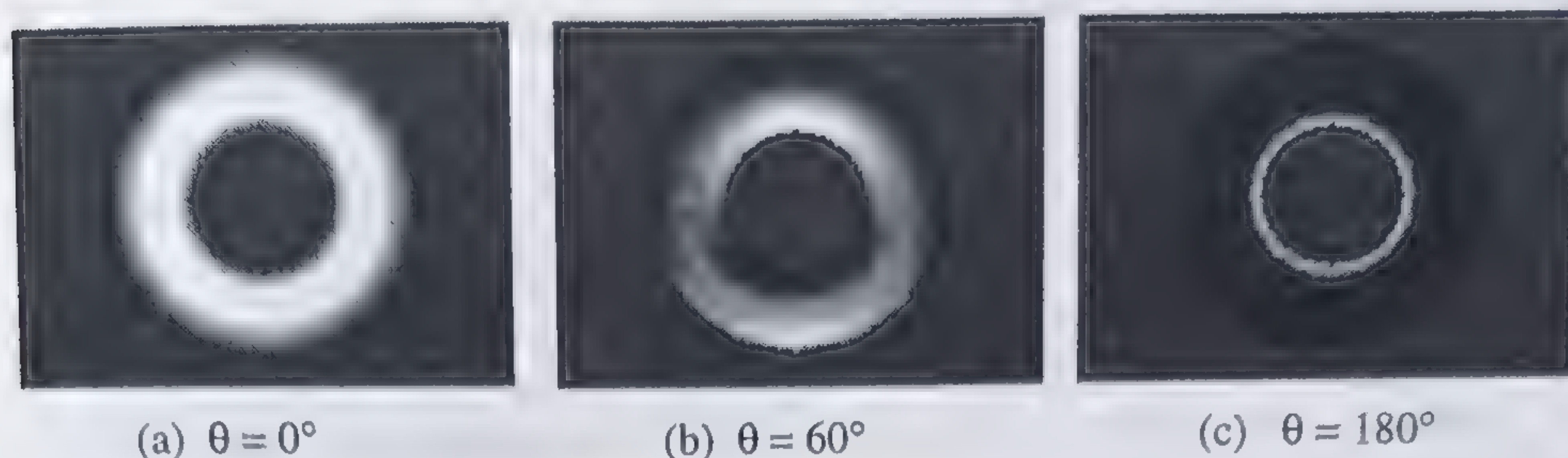


Fig. 7.16 Sursă de lumină care se deplasează în scenă.

### 7.5.5 COMBINAREA CULORILOR

Mai multe efecte în redarea obiectelor (transparență, anti-aliasing, ceață) se obțin prin combinarea culorilor la nivel de pixel.

Pentru validarea operației de combinare a culorilor la nivel de pixel (*blending*) se apelează funcția `glEnable(GL_BLEND)`. În cursul combinării, culoarea unui nou fragment (sursa) este combinată cu culoarea memorată într-o locație corespunzătoare din bufferul de cadru, care este și destinația pentru înscriserea rezultatului combinării.

Pentru combinarea culorilor se calculează factorii de combinare ai sursei și ai destinației. Acești factori sunt cvadrupleți RGBA, care se multiplică cu componentele corespunzătoare R, G, B și A ale sursei, respectiv ale destinației. În tabelul 7.2 sunt date valorile constantelor simbolice predefinite în OpenGL, din care se pot selecta factorii de combinare ai sursei și ai destinației.

Tabelul 7.2

Factorii de combinare ai sursei și ai destinației

Constanta	Sursă-destinație	Factorul de combinare
GL_ZERO	sursă sau dest	(0,0,0,0)
GL_ONE	sursă sau dest	(1,1,1,1)
GL_DST_COLOR	sursă	$(R_d, G_d, B_d, A_d)$
GL_SRC_COLOR	destinație	$(R_s, G_s, B_s, A_s)$
GL_ONE_MINUS_DST_COLOR	sursă	$(1,1,1,1) - (R_d, G_d, B_d, A_d)$
GL_ONE_MINUS_SRC_COLOR	destinație	$(1,1,1,1) - (R_s, G_s, B_s, A_s)$
GL_SRC_ALPHA	sursă sau dest	$(A_s, A_s, A_s, A_s)$
GL_DST_ALPHA	sursă sau dest	$(A_d, A_d, A_d, A_d)$
GL_ONE_MINUS_SRC_ALPHA	sursă sau dest	$(1,1,1,1) - (A_s, A_s, A_s, A_s)$
GL_ONE_MINUS_DST_ALPHA	sursă sau dest	$(1,1,1,1) - (A_d, A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	sursă	$(f, f, f, 1); f = \min(A_s, 1 - A_d)$



Fie factorii de combinare ai sursei ( $S_r, S_g, S_b, S_a$ ) și ai destinației ( $D_r, D_g, D_b, D_a$ ) și componentele culorii sursei ( $R_s, G_s, B_s, A_s$ ) și ale destinației, ( $R_d, G_d, B_d, A_d$ ). Culoarera rezultată prin combinare, care se înscrie în locația de destinație, are componentele ( $R_s S_r + R_d D_r, G_s S_g + G_d D_g, B_s S_b + G_d D_g, A_s S_a + A_d D_a$ ). Este posibil ca fiecare componentă a culorii rezultate să fie limitată la intervalul  $[0,1]$ .

Factorii de combinare se pot selecta din mai multe valori posibile prin argumentele transmise funcției:

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
```

#### 7.5.5.1 Transparența

Transparența suprafețelor se implementează folosind combinarea cu factorii de combinare `GL_SRC_ALPHA`, pentru sursă și `GL_ONE_MINUS_SRC_ALPHA` pentru destinație.

#### ■ Exemplul 7.7

În programul următor se suprapun parțial suprafețe transparente triunghiulare de culoare gri deschis (0.8, 0.8, 0.8) peste un dreptunghi de culoare gri închis (0.2, 0.2, 0.2).

În continuare sunt prezentate funcțiile callback `Init()` și `Display()` ale programului dezvoltat sub GLUT. Celelalte funcții, `main()` și `Reshape()` nu au nimic special, fiind asemănătoare cu cele din programele precedente.

```
void Init(){
    glClearColor(1.0,1.0,1.0,1.0);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}
void Triangle() {
    glBegin(GL_POLYGON);
        glVertex3d(-2,-2,0);
        glVertex3d( 2,-2,0);
        glVertex3d( 0, 2,0);
    glEnd();
}
void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslated(0.0,0.0,-20.0); // transf. de observare
    glDisable(GL_BLEND);
    glColor3d(0.2,0.2,0.2);
    glBegin(GL_POLYGON);           // desenare dreptunghi
        glVertex3d(-8,0,0);
        glVertex3d( 8,0,0);
        glVertex3d( 8,4,0);
        glVertex3d(-8,4,0);
    glEnd();
    glEnable (GL_BLEND);
```



```

glPushMatrix();
    glTranslated(-6.0,4.0,0.0);
    glColor4d(0.8,0.8,0.8,0.25);
    Triangle(); // primul triunghi
glPopMatrix();
glPushMatrix();
    glTranslated(-2.0,4.0,0.0);
    glColor4d(0.8,0.8,0.8,0.50);
    Triangle(); // al doilea triunghi
glPopMatrix();
glPushMatrix();
    glTranslated(2.0,4.0,0.0);
    glColor4d(0.8,0.8,0.8,0.75);
    Triangle(); // al treilea triunghi
glPopMatrix();
glPushMatrix();
    glTranslated(6.0,4.0,0.0);
    glColor4d(0.8,0.8,0.8,1.0);
    Triangle(); // al patrulea triunghi
glPopMatrix();

glPopMatrix();
glutSwapBuffers();
}

```

În funcția `Init()` se setează culoarea de ștergere albă (1.0, 1.0, 1.0, 1.0) a bufferului de culoare și factorii de combinare `GL_SRC_ALPHA`, pentru sursă și `GL_ONE_MINUS_SRC_ALPHA` pentru destinație.

Imaginea din fig. 7.16 s-a obținut pentru valorile 0.25, 0.50, 0.75, 1.0 ale componentei  $A_s$  a triunghiului. Combinarea culorii unei noi suprafețe cu culoarea existentă în buffer produce aspectul de transparență a suprafețelor.

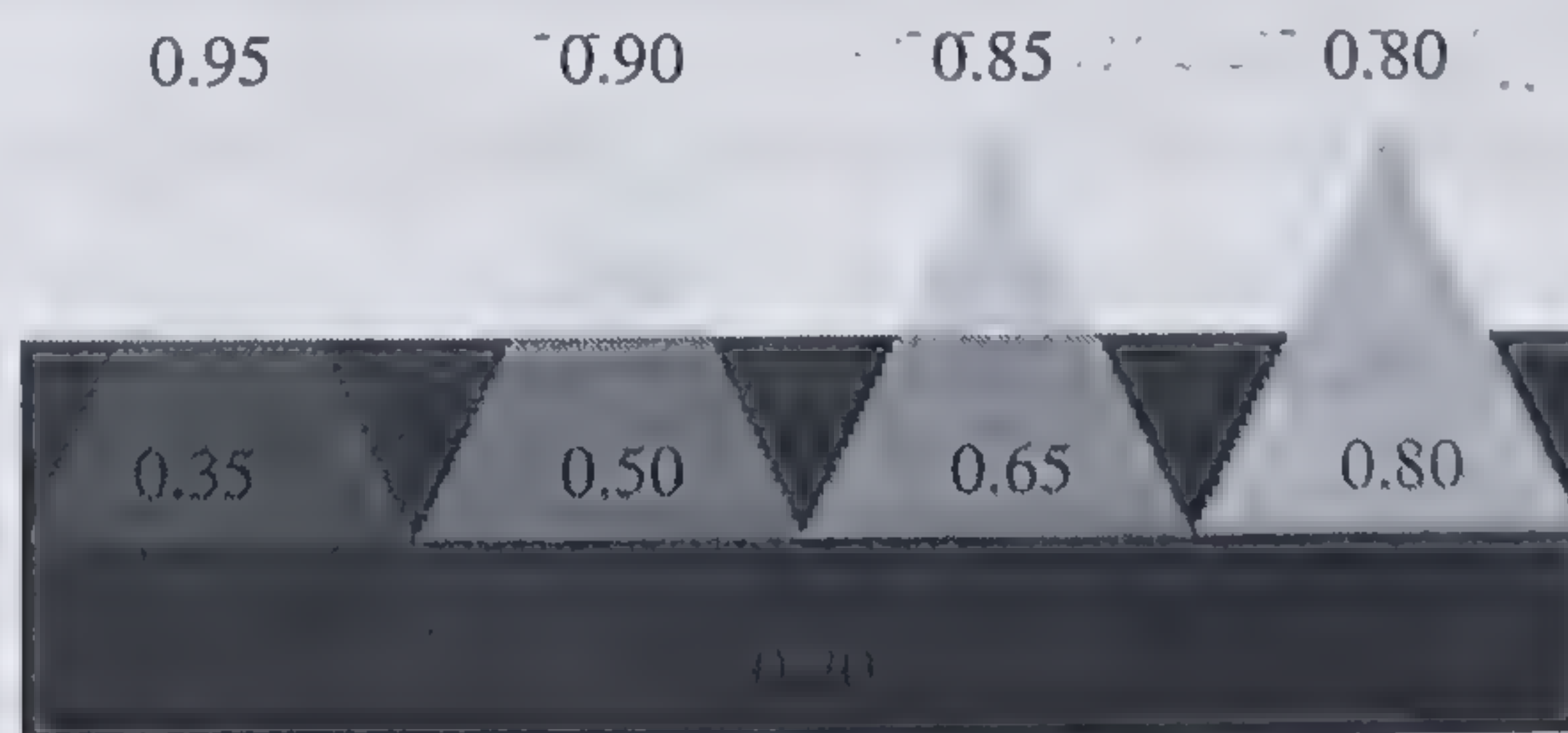


Fig. 7.16 Combinarea culorilor în calculul transparenței.

Dreptunghiul se înscrie în buffer cu culoarea lui inițială. Culoarea fiecăruia din cele 4 triunghiuri care urmează (poligoane sursă) se combină la nivel de pixel cu culoarea existentă în buffer (care poate fi culoarea de ștergere sau culoarea dreptunghiului) folosind pentru factorii de combinare valoarea transparenței sale.



Primele trei triunghiuri sunt compuse din două culori: una rezultată din combinarea culorii proprii cu culoarea dreptunghiului, cealaltă rezultată din combinarea culorii proprii cu culoarea de ștergere. De exemplu, pentru primul triunghi, în partea de jos, suprapusă peste dreptunghi, fiecare componentă de culoare R,G,B are intensitatea  $0.25 \cdot 0.8 + (1 - 0.25) \cdot 0.2 = 0.35$ ; în partea de sus, prin combinare cu culoarea de ștergere, fiecare componentă de culoare R,G,B rezultată are intensitatea  $0.25 \cdot 0.8 + (1 - 0.25) \cdot 1.8 = 0.95$ . Pentru al doilea triunghi componentele de culoare sunt 0.5 pentru partea de jos și 0.9 pentru partea de sus; pentru al treilea triunghi componentele de culoare sunt 0.65 pentru partea de jos și 0.85 pentru partea de sus. Ultimul triunghi, cu  $A_s = 1$ , înlocuiește complet valorile existente în buffer și este desenat cu componentele de culoare 0.8 în ambele părți de suprapunere.

De remarcat că în acest program simplu nu este validat bufferul de adâncime. În mod implicit, în OpenGL testul de adâncime este invalidat și, deoarece nu se validează nici la inițializare nici în altă parte în program, testul de adâncime nu se efectuează și poligoanele sunt înscrise în bufferul de imagine în ordinea în care sunt transmise, indiferent de adâncimea lor.

Acest fapt trebuie să fie evitat atunci când sunt reprezentate obiecte tridimensionale opace și transparente în scenă. Dat fiind că obiectele tridimensionale se pot ascunde unele pe altele este necesar să se valideze testul de adâncime prin apelul funcției `glEnable(GL_DEPTH_TEST)`. Obiectele opace se desenează mai întâi și se execută operațiile cu bufferul de adâncime în mod normal. Pentru desenharea obiectelor transparente, se setează condiția de acces numai pentru citire a bufferului de adâncime (read-only) prin apelul funcției `glDepthMask(GL_FALSE)`. Atunci când se desenează obiecte transparente, adâncimea lor se compară cu adâncimile memorate în Z-buffer, stabilite de obiectele opace, astfel că nu se desenează părțile ascunse (aflate în spatele) obiectelor opace. Dacă sunt mai aproape de punctul de observare, se desenează prin combinarea culorii, dar nu se elimină adâncimea obiectelor opace din bufferul de adâncime, deoarece acesta este read-only.

În Planșa 5 (stânga) sunt reprezentate două suprafețe colorate suprapuse cu transparență.

#### 7.5.5.2 Simularea ceții

Calculul ceții se validează prin apelul funcției `glEnable(GL_FOG)`. Se pot defini trei moduri de calcul af funcției de combinare  $f$ , corespunzător unei variații exponențiale, pătratic exponențiale și liniare. Funcțiile prin care se pot defini parametrii de calcul ai ceții sunt:

```
void glFogf(GLenum pname, GLfloat param);
void glFogi(GLenum pname, GLint param);
void glFogfv(GLenum pname, const GLfloat *params);
void glFogiv(GLenum pname, const GLint *params);
```

Parametrul `pname` poate fi una din valorile `GL_FOG_DENSITY`, `GL_FOG_MODE`, `GL_FOG_START`, `GL_FOG_END` pentru definirea modului de



calcul, a densității, a distanței de început și a distanței de sfârșit a ceții. Dacă se definește modul de calcul, atunci parametrul `param` poate lua una din constantele simbolice `GL_LINEAR`, `GL_EXP`, `GL_EXP2`. Dacă se definește densitatea, atunci `param` va avea o valoare pozitivă care este folosită în calculele exponențiale. Distanța de început sau de sfârșit a ceții, folosită în funcția liniară de calcul, se transmite prin argumentul `param` corespunzător parametrului `GL_FOG_START`, respectiv `GL_FOG_END`.

Pentru a defini culoarea ceții se apelează una din funcțiile `glFogiv()` sau `glFogfv()` în care argumentul `pname` are valoarea `GL_FOG_COLOR`, iar argumentul `params` este un pointer la un vector de patru valori întregi sau numere în virgulă flotantă, care conțin componentele R,G,B,A de culoare ale ceții.

### ■ Exemplul 7.8

În acest exemplu se desenează cinci obiecte la distanțe diferite. Dat fiind că se validează calculul ceții, obiectele se văd din ce în ce mai estompate, culoarea lor apropiindu-se de culoarea de ștergere cu care se face combinarea.

```
#include <GL/glut.h>
static GLuint listName;
void Teapot (GLfloat x, GLfloat y, GLfloat z){
    glPushMatrix();
    glTranslatef(x, y, z);
    glutSolidTeapot(0.9);
    glPopMatrix();
}
void Init(){
    float fogDensity = 0.04;
    float fog_color[] = {0.5, 0.5, 0.5, 1.0};
    float mat_ambient[] = {0.2, 0.0, 0.0, 1.0};
    float mat_diffuse[] = {0.6, 0.0, 0.0, 1.0};
    float mat_shininess[] = {90.0};
    float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glEnable(GL_FOG);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glFogi(GL_FOG_MODE, GL_EXP);
    glFogf(GL_FOG_DENSITY, fogDensity);
    glFogfv(GL_FOG_COLOR, fog_color);
    glClearColor(0.5, 0.5, 0.5, 1.0);
    listName = glGenLists(1);
    glNewList(listName, GL_COMPILE);
```



```

        Teapot (-3.0, -0.5, -10.0);
        Teapot (-2.0, -0.5, -20.0);
        Teapot (0.0, -0.5, -30.0);
        Teapot (2.5, -0.5, -40.0);
        Teapot (5.0, -0.5, -50.0);
    glEndList();
}

void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glCallList(listName);
    glPopMatrix();
    glutSwapBuffers();
}

void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 0.1, 4000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
                        GLUT_DEPTH);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simularea cetii");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```

Programul creează o listă de display care conține apelurile funcțiilor OpenGL pentru generarea imaginii. La execuția acestui program se obține imaginea din Planșa 5 (dreapta).



## MODELAREA ȘI REDAREA SUPRAFETELOR PARAMETRICE

Cele mai utilizate tipuri de curbe și suprafețe parametrice în proiectarea grafică sunt curbele și suprafețele Bézier și B-spline, care vor fi descrise în continuare.

### 8.1 CURBE BÉZIER

Cea mai cunoscută formă de reprezentare parametrică a curbelor și suprafețelor este cea dezvoltată de Pierre Bézier pentru proiectarea caroseriei mașinilor Renault [Bez72]. Sistemul de proiectare UNISURF, bazat pe reprezentarea parametrică a suprafețelor, a fost folosit pentru proiectarea multor modele de mașini Renault.

Curbele Bézier sunt curbe parametrice cubice în care se folosesc patru puncte de control,  $P_0, P_1, P_2, P_3$ , iar funcțiile de bază sunt funcțiile Bézier, care au următoarele expresii:

$$\begin{aligned} B_0(u) &= (1-u)^3 \\ B_1(u) &= 3u(1-u)^2 \\ B_2(u) &= 3u^2(1-u) \\ B_3(u) &= 3u^3 \end{aligned} \quad (8.1)$$

Rezultă expresia polinomială a curbelor Bézier:

$$Q(u) = P_0(1-u)^3 + 3P_1u(1-u)^2 + 3P_2u^2(1-u) + 3P_3u^3 \quad (8.2)$$

Punctele  $P_0, P_1, P_2, P_3$  se numesc puncte de control deoarece poziția lor în spațiu influențează forma curbei.  $P_0$  și  $P_3$  sunt capetele curbei. Poligonul obținut prin unirea punctelor succesive se numește *poligon caracteristic* sau *poligon de control*. În sensul definiției date poligonului în secțiunea precedentă, poligonul caracteristic ar trebui închis cu ultimul segment, de la  $P_3$  la  $P_0$ ; în general, însă, acest segment al poligonului caracteristic nu se reprezintă. În fig. 8.1 (a) sunt



arătate trei curbe Bézier cubice, cu poligoanele caracteristice. Programul care generează această imagine este descris în exemplul 8.1.

Vectorii tangenți la curba Bézier se obțin prin derivarea funcțiilor de bază în raport cu parametrul  $u$  și au valorile în capete:

$$Q_u(0) = 3(P_1 - P_0)$$

$$Q_u(1) = 3(P_2 - P_3)$$

Din aceste relații rezultă că punctele  $P_0$  și  $P_1$  se află pe tangenta la curbă în punctul  $P_0$ , iar punctele  $P_2$  și  $P_3$  se află pe tangenta la curbă în punctul  $P_3$ .

Reprezentarea grafică a celor patru funcții de bază Bézier cubice (fig. 8.1(b)) arată în ce proporție influențează un punct de control forma curbei Bézier. Punctul de control  $P_0$  are cea mai mare influență la  $u = 0$ , unde  $B_0 \doteq 1$ , iar celelalte funcții,  $B_1, B_2, B_3$ , sunt 0. Punctul de control  $P_3$  are cea mai mare influență la  $u = 1$ , unde  $B_3 = 1$ , iar celelalte funcții,  $B_0, B_1, B_2$ , sunt egale cu zero. Punctele  $P_1$  și  $P_2$  au cel mai mare efect la  $u \doteq 1/3$ , respectiv  $u = 2/3$ . Se poate remarca faptul că modificarea fiecărui punct de control influențează, cu un efect mai mare sau mai mic, forma întregii curbe Bézier. Acesta este, de fapt, dezavantajul principal al curbelor Bézier: controlul global al formei curbei.

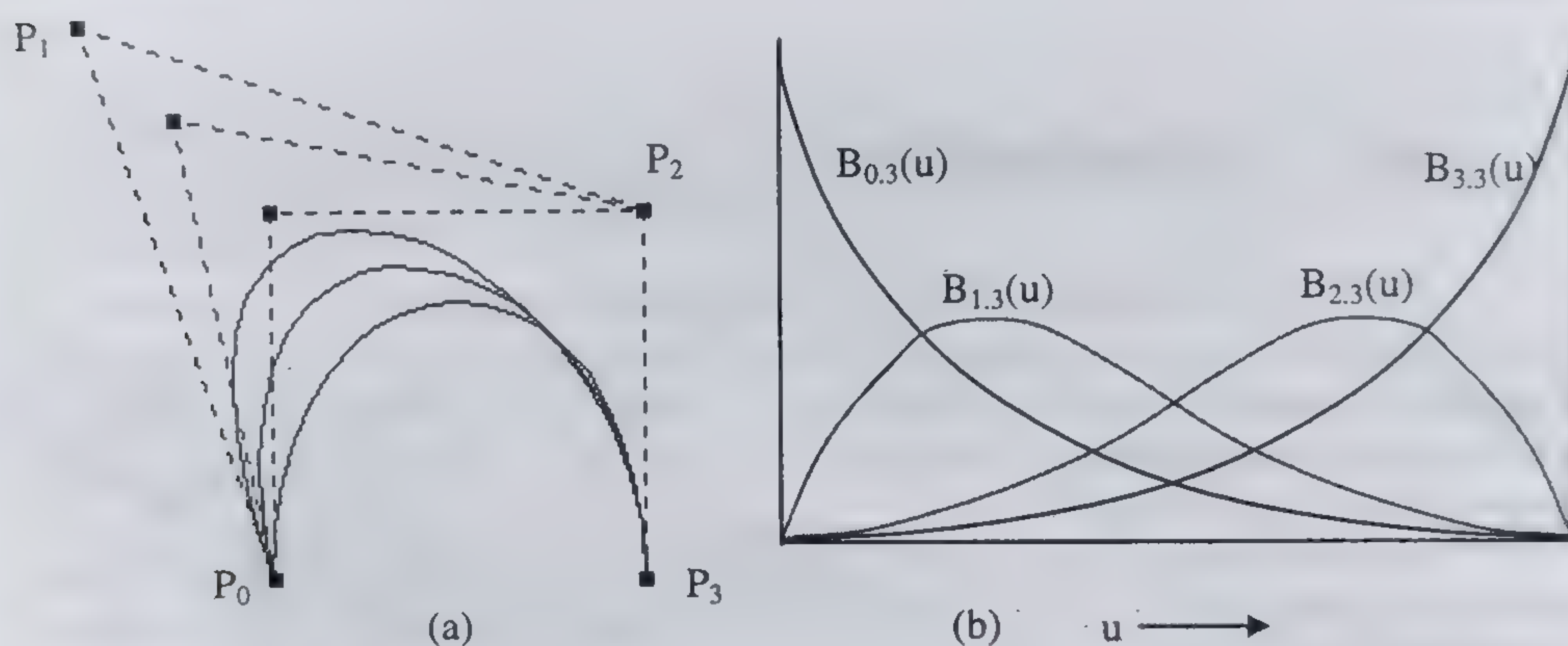


Fig. 8.1 (a) Curbe Bézier cubice și poligoanele caracteristice.  
(b) Funcțiile de bază Bézier cubice.

Funcțiile de bază Bézier cubice exprimate de relația (2.10), pot fi considerate un caz particular al unor funcții de bază de un ordin oarecare  $n$ , pentru exprimarea parametrică a unei curbe sub forma:

$$Q(u) = \sum_{i=0}^n P_i B_{i,n}(u) \quad (8.3)$$

unde  $B_{i,n}$  sunt polinoame Bernstein;

$$B_{i,n}(u) = C(n, i) u^i (1-u)^{n-i} \quad (8.4)$$

cu coeficienții  $C(n, i) = n! / (i!(n-i)!)$ .



Pentru funcții de bază de gradul  $n$  sunt necesare  $n+1$  puncte de control; numărul punctelor de control dă ordinul curbei Bézier. Funcțiile de bază de ordin mai mare de trei permit descrierea unor curbe mai complexe, dar timpul de calcul al punctelor de pe curbă este mai ridicat (sunt necesare mai multe operații pentru fiecare punct) și, de aceea, sunt rar utilizate în grafica interactivă sau în modelare. În plus, cu cât crește gradul funcțiilor de bază, relația dintre poligonul caracteristic și forma curbei devine mai slabă și, de aceea, proiectantul controlează mai greu forma curbei.

Curbe mai complexe se pot obține și prin combinarea mai multor segmente de curbe Bézier cubice. Combinarea mai multor segmente necesită impunerea unor condiții care să asigure continuitatea segmentelor. Condiția de continuitate pozițională (de ordinul 0) a segmentelor de curbă este ca punctul de control final al unui segment să fie primul punct de control al segmentului următor (fig.8.2(a)). Această condiție nu este suficientă, deoarece pot să apară discontinuități la redarea obiectelor. Condiția de continuitate tangențială (de ordinul 1) este ca segmentele de curbă să aibă aceeași tangentă în punctul de alipire. Aceasta înseamnă că este necesar ca muchiile adiacente a celor două poligoane caracteristice să fie coliniare (fig. 8.2(b)).

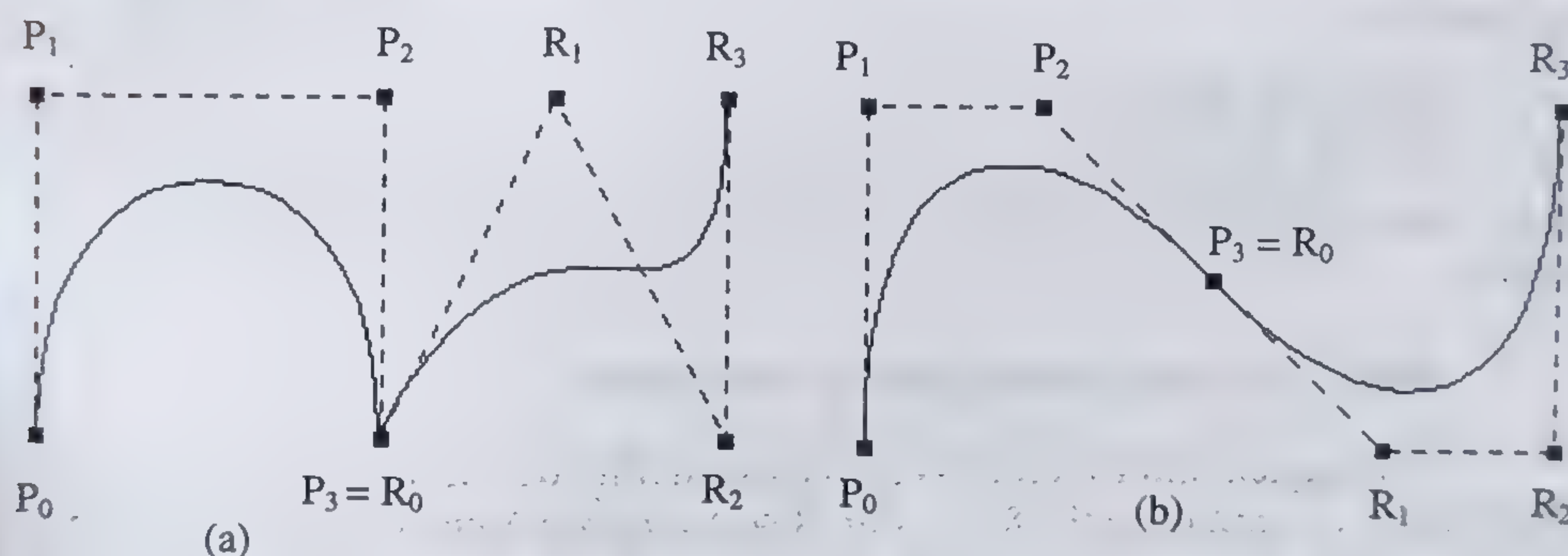


Fig. 8.2 Continuitatea între două segmente de curbă Bézier.  
(a) Continuitate pozițională (b) Continuitate tangențială.

Folosind continuitatea tangențială se pot construi curbe din mai multe segmente, dacă proiectatul are în vedere ca un nou punct de control adăugat să fie coliniar cu ultimele două (de exemplu,  $R_1$  trebuie să fie coliniar cu  $P_3$  și  $P_2$ ). Acest dezavantaj nu apar la definirea curbelor B-spline.

## 8.2 CURBE B-SPLINE'

Două din dezavantajele curbelor Bézier, și anume nelocalitatea controlului și relația dintre numărul punctelor de control și gradul curbei, sunt înlăturate în reprezentarea parametrică a curbelor B-spline.



O curbă B-spline poate fi compusă din oricâte segmente cubice pe porțiuni, cu schimbarea corespunzătoare a coeficienților la trecerea de la un segment la altul. Pentru o secvență de puncte de control  $P_i$ , ecuația segmentului  $Q_i(u)$  al curbei B-spline este:

$$Q_i(u) = \sum_{k=0}^3 P_{i-3+k} B_{i-3+k}(u) \quad (8.5)$$

unde  $i$  este numărul segmentului de curbă, iar  $k$  este indexul punctului de control local, adică este indexul pentru segmentul  $i$ . Folosind această notație se poate considera  $u$  un parametru local variind în intervalul închis  $[0,1]$  pentru definirea segmentului dat.

O curbă B-spline este compusă din  $m - 2$  segmente notate  $Q_3, Q_4, \dots, Q_m$ , controlate de  $m + 1$  puncte de control  $P_0, P_1, P_2, \dots, P_m$ , unde  $m \geq 3$ . Fiecare segment al curbei este definit de patru puncte de control și fiecare punct de control influențează numai patru segmente din curbă. Aceasta este proprietatea de control local a curbelor B-spline, care este un avantaj față de curbele Bézier.

Un segment de curbă B-spline prezintă continuitate de ordinul 0, 1 și 2 și se poate defini o mulțime de segmente de curbă B-spline ca o singură curbă cu parametrul  $u$ :

$$Q(u) = \sum_{i=0}^m P_i B_i(u) \quad (8.6)$$

unde  $i$  este numărul unui punct de control nelocal, iar  $u$  este parametrul global al curbei.

### 8.2.1 CURBE B-SPLINE UNIFORME

Un segment dintr-o curbă B-spline este definit de patru puncte de control și patru funcții de bază (ecuația 8.5). Fig. 8.3(a) reprezintă un segment de curbă B-spline definită de patru puncte de control  $P_0, P_1, P_2, P_3$ . Se observă că, spre deosebire de curbele Bézier, curbele B-spline uniforme nu interpolează punctele de control de la capete. Fig. 8.3 (b) arată o curbă B-spline definită de șase puncte de control  $P_0, P_1, P_2, P_3, P_4, P_5$ , formată din trei segmente de curbă, notate  $Q_3, Q_4, Q_5$ . Capătul stâng al segmentului  $Q_3$  se află în vecinătatea punctului de control  $P_0$ ; capătul dreapta al segmentului  $Q_5$  se află în vecinătatea punctului de control  $P_5$ .

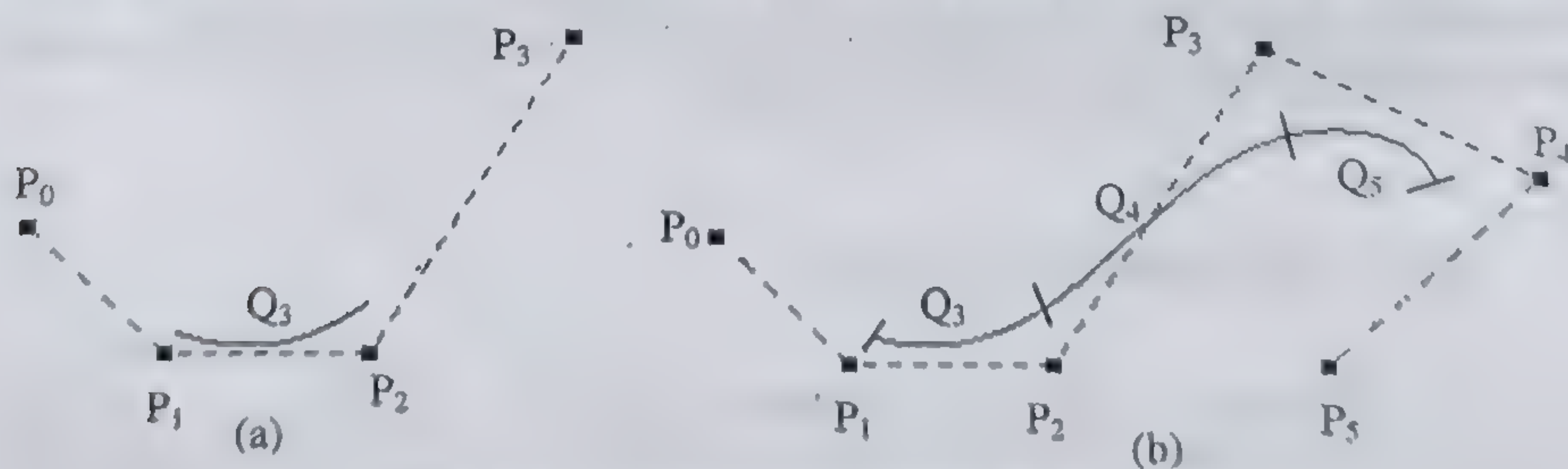


Fig. 8.3 (a) Segment de curbă B-spline.  
(b) Curbă B-spline uniformă compusă din trei segmente.



Folosind notația anterioară, se poate descrie organizarea segmentelor curbei din fig. 8.3 (b) astfel:

$Q_3$  este definit de punctele  $P_0P_1P_2P_3$  și funcțiile  $B_0B_1B_2B_3$

$Q_4$  este definit de punctele  $P_1P_2P_3P_4$  și funcțiile  $B_1B_2B_3B_4$

$Q_5$  este definit de punctele  $P_2P_3P_4P_5$  și funcțiile  $B_2B_3B_4B_5$

Faptul că segmentele de curbă adiacente au trei puncte de control comune asigură continuitatea de ordin 0, 1 și 2. Funcțiile de bază B-spline sunt diferite de zero în patru intervale succesive,  $u_i, u_{i+1}, u_{i+2}, u_{i+3}$ , centrate la  $u_{i+2}$  (fig. 8.4).

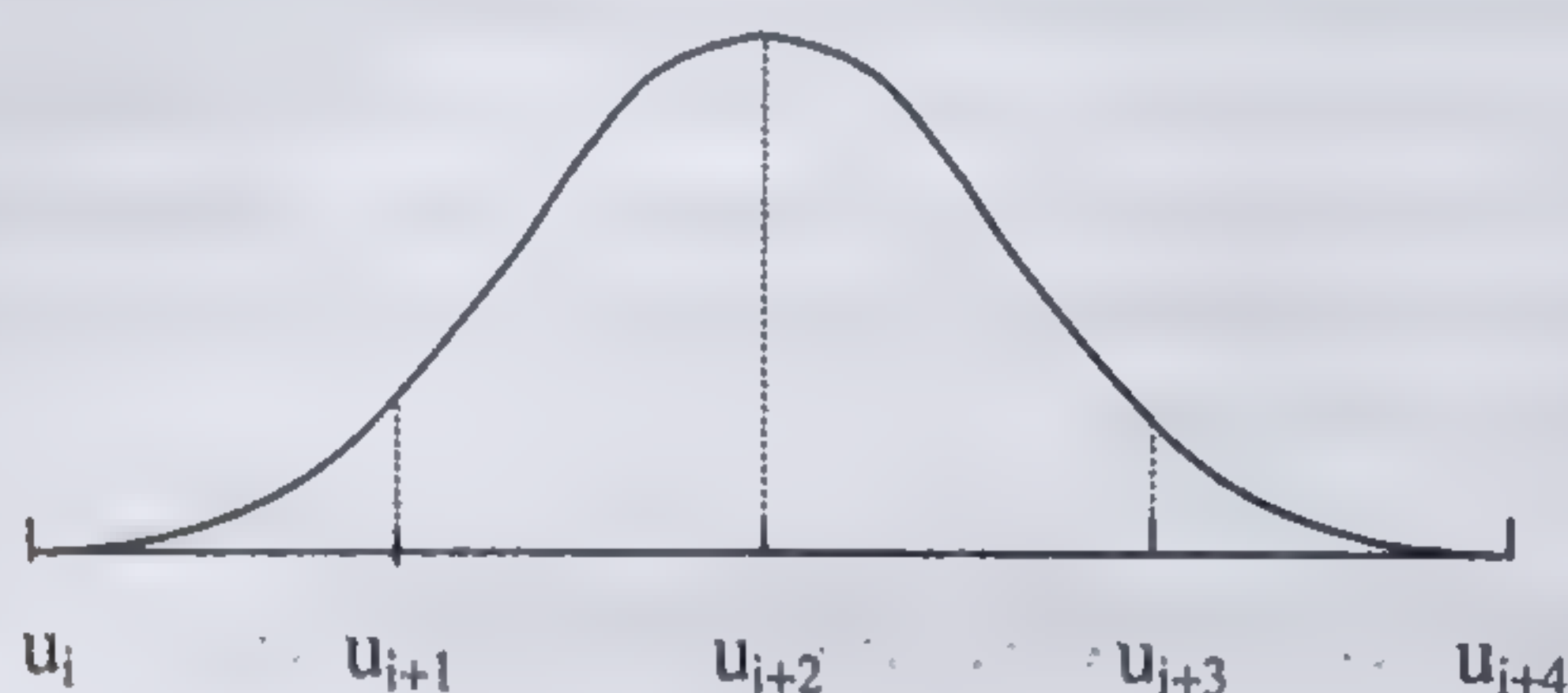


Fig. 8.4 Funcția B-spline  $B_i(u)$ .

Fiecare funcție de bază B-spline uniformă este o copie translatată a unei astfel de funcții. În intervalul de la  $u_i$  la  $u_{i+1}$  în care este definit un segment de curbă sunt active patru funcții de bază, iar suma lor este egală cu 1 (fig. 8.5).

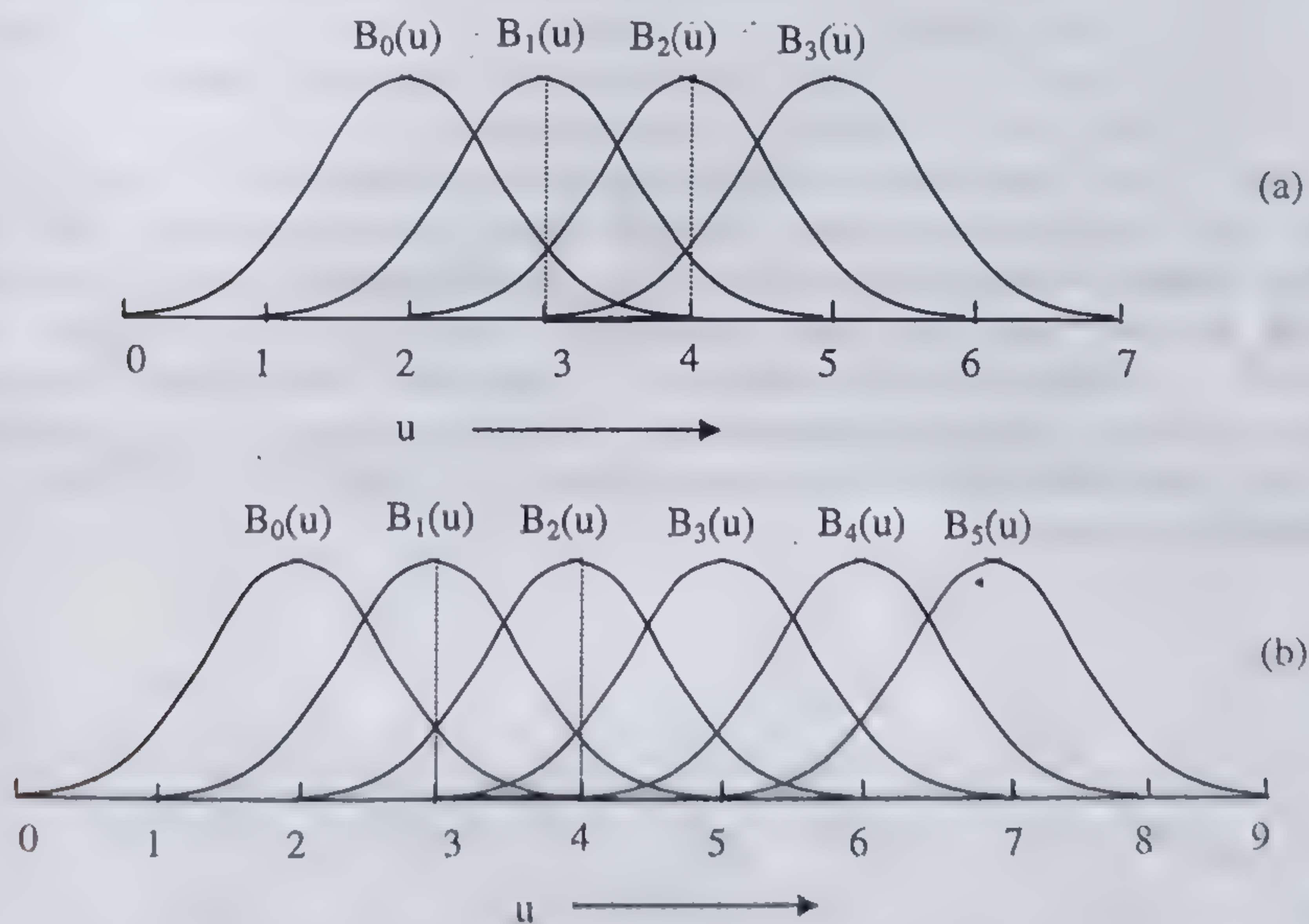


Fig. 8.5 Funcțiile de bază B-spline uniforme pentru un segment (a) și pentru o curbă formată din trei segmente (b).



Punctele de adiacență între funcțiile de bază se numesc *noduri* (*knots*) și ele se specifică prin valoarea parametrului  $u$  la care o funcție de bază devine inactivă și alta devine activă. Pentru un segment de curbă sunt folosite patru funcții de bază și fiecare funcție este activă pe patru intervale ale parametrului  $u$ , deci un segment este definit peste 8 noduri (fig. 8.5(a)). O curbă B-spline la care intervalele între noduri sunt egale se numește curbă B-spline uniformă.

Se poate rezuma că o curbă B-spline este compusă din  $m - 2$  segmente, definite de  $m + 1$  funcții de bază, peste  $m + 5$  noduri. Pentru curba din fig. 8.3(b),  $m = 5$ , deci sunt 6 puncte de control și 6 funcții de bază, 3 segmente de curbă și 10 noduri, distribuite la intervale egale ale parametrului  $u$ .

Dezavantajul curbelor B-spline uniforme este acela că nu interpolează punctele de control de la capete, iar adăugarea unor puncte de control multiple reduce continuitatea între segmente. Rezultate de interpolare mai bune se obțin folosind curbe B-spline neuniforme.

### 8.2.2 CURBE B-SPLINE NEUNIFORME

O curbă B-spline neuniformă este o curbă în care intervalele parametrice între noduri succesive nu sunt neapărat egale. Acest lucru înseamnă că funcțiile de amestec nu mai sunt translatate una față de cealaltă, ci variază de la un interval la altul. Forma obișnuită a curbelor B-spline neuniforme este aceea în care unele intervale între noduri sunt zero, adică nodurile sunt multiple. Acest lucru permite interpolarea tuturor punctelor de control (capete sau intermediare).

Curbele B-spline din fig. 8.3 sunt curbe uniforme, deoarece nodurile sunt dispuse uniform pe axa parametrului  $u$ . Așa cum reiese și din fig. 8.5, valorile nodurilor sunt 0, 1, 2, 3, 4, 5, 6, 7 pentru un segment de curbă și, respectiv 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 pentru curba formată din trei segmente.

Dacă se modifică poziția nodurilor pe axa parametrului  $u$ , astfel încât intervalele între acestea să nu mai fie toate egale, se obțin curbele B-spline neuniforme. Segmentul de curbă din fig. 8.6 (a) este un segment de curbă B-spline neuniformă în care nodurile sunt dispuse la valorile  $[0, 0, 0, 0, 1, 1, 1, 1]$  ale parametrului  $u$ . Multiplicitatea nodurilor la capetele segmentului asigură interpolarea capetelor poligonului caracteristic. În mod asemănător, curba B-spline din fig. 8.6(b) este neuniformă, cu vectorul de noduri:  $[0, 0, 0, 0, 1, 2, 3, 3, 3, 3]$ , deci cu noduri multiple la capete.

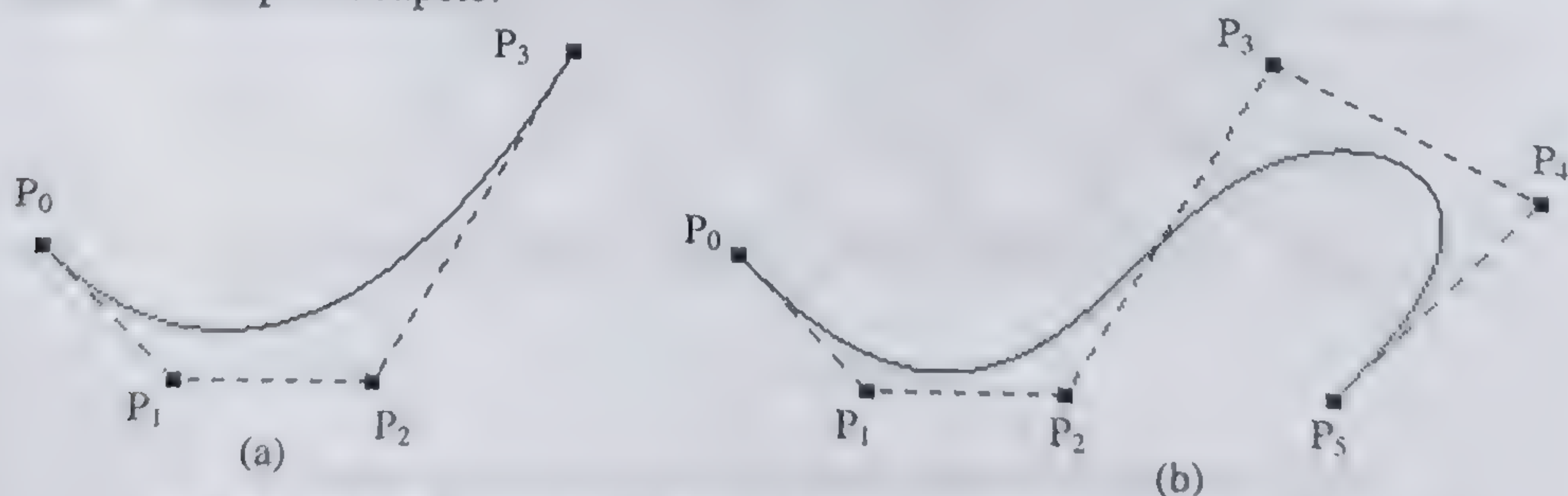


Fig. 8.6 (a) Segment de curbă B-spline neuniformă cu nodurile  $[0, 0, 0, 0, 1, 1, 1, 1]$ .  
(b) Curbă B-spline neuniformă cu nodurile  $[0, 0, 0, 0, 1, 2, 3, 3, 3, 3]$ .



Algoritmul de calcul al funcțiilor de bază B-spline (cunoscut sub numele de algoritmul *Cox-deBoor*) generează recursiv funcțiile de bază B-spline uniforme sau neuniforme de orice grad. Dacă se definește  $B_{i,j}(u)$  funcția de bază de ordinul  $j$  care ponderează punctul  $P_i$ , (ordinul este gradul polinomului plus 1), atunci funcțiile de bază B-spline cubice (de ordin 4) se pot calcula astfel:

$$\begin{aligned}
 B_{i,1}(u) &= \begin{cases} 1 & \text{pentru } u_i \leq u \leq u_{i+1} \\ 0 & \text{altfel} \end{cases} \\
 B_{i,2} &= \frac{u - u_i}{u_{i+1} - u_i} B_{i,1}(u) + \frac{u_{i+2} - u}{u_{i+2} - u_{i+1}} B_{i+1,1}(u) \\
 B_{i,3} &= \frac{u - u_i}{u_{i+2} - u_i} B_{i,2}(u) + \frac{u_{i+3} - u}{u_{i+3} - u_{i+1}} B_{i+1,2}(u) \\
 B_{i,4} &= \frac{u - u_i}{u_{i+3} - u_i} B_{i,3}(u) + \frac{u_{i+4} - u}{u_{i+4} - u_{i+1}} B_{i+1,3}(u) \quad (8.7)
 \end{aligned}$$

În formulele Cox-deBoor, poate apare împărțirea 0/0, atunci când un nod se repetă; algoritmul atribuie rezultat zero unei astfel de împărțiri prin testarea numărătorului și, dacă acesta este zero, rezultatul este zero, indiferent de valoarea numitorului.

Multiplicarea nodurilor are un efect de asimetrizare a funcțiilor de bază și acest lucru dă posibilitatea ca un segment de curbă să fie obligat să treacă printr-un punct de control dat. În fig. 8.7 sunt date curbele B-spline care se obțin folosind aceleași puncte de control, pentru diferite valori ale nodurilor.

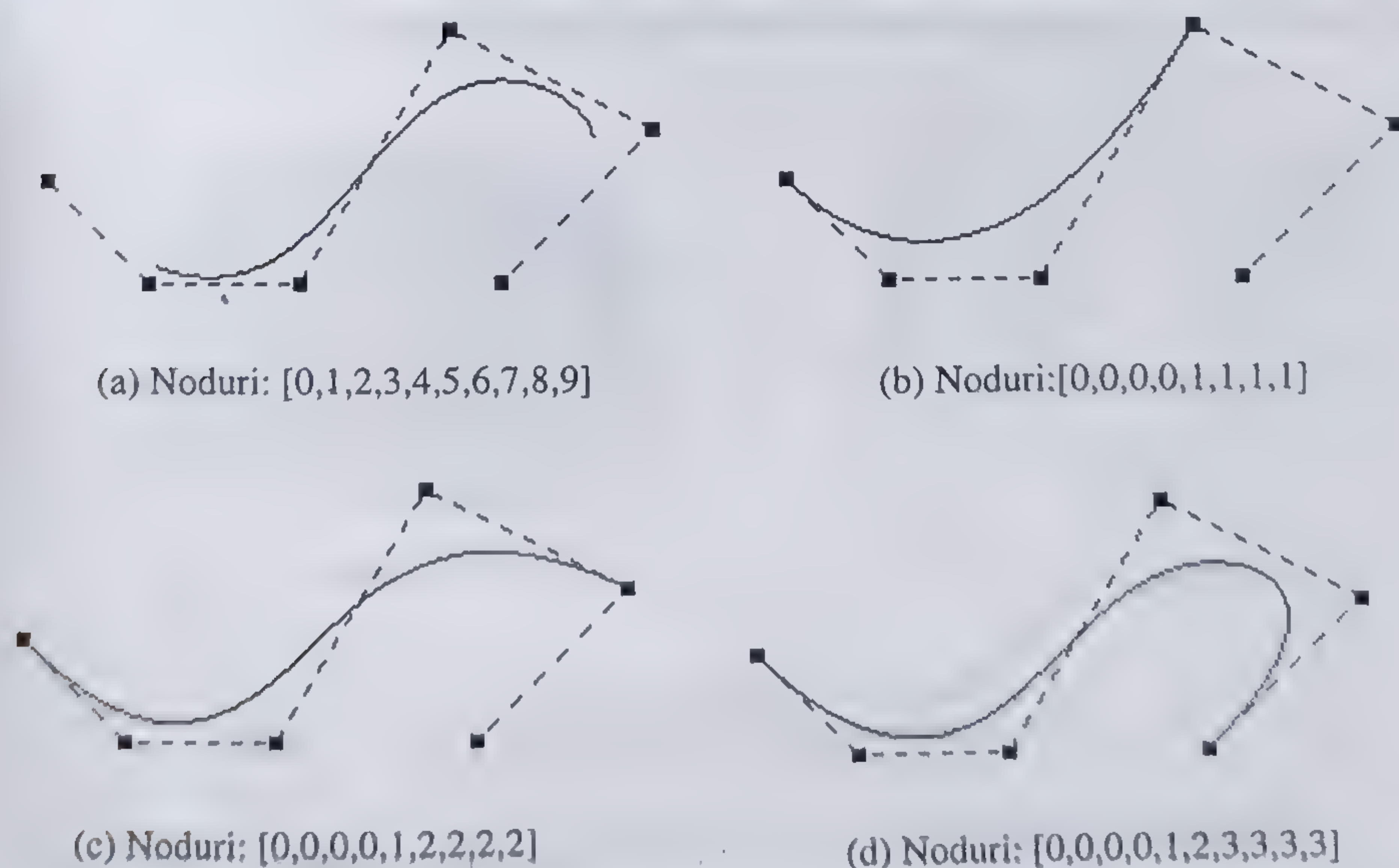


Fig. 8.7 Curbe B-spline pentru diferite valori ale nodurilor.



În primul caz (a), nodurile sunt distribuite uniform și curba este B-spline uniformă, care nu interpolează punctele terminale. În al doilea caz, s-a obținut un singur segment care interpolează între  $P_0$  și  $P_3$  și care este o curbă Bezier. Deci, atât curbele B-spline uniforme, cât și curbele Bezier sunt cazuri particulare ale curbelor B-spline neuniforme. În cazurile (c) și (d) nodurile multiple asigură interpolarea între anumite puncte de control. Acestea sunt curbe B-spline neuniforme.

Curbele B-spline neuniforme sunt deosebit de flexibile, permițând reprezentarea unor forme complexe cu un număr relativ mic de funcții de bază de grad fix (cubic în mod obișnuit).

### 8.3 SUPRAFETE BÉZIER

Se poate extinde reprezentarea parametrică cubică a segmentelor de curbă, descrisă în subcapitolele precedente, la reprezentarea peticelor de suprafețe parametrice bicubice. Un petic de suprafață bicubică Bézier este definit prin ecuațiile:

$$Q(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 P_{ij} B_i(u) B_j(v) \quad (8.8)$$

Un petic Bézier este definit prin 16 puncte de control, care formează o suprafață de control numită poliedru caracteristic. Denumirea care se dă acestei suprafețe este oarecum improprie, dat fiind că este nu este o suprafață închisă, așa cum este definit un poliedru. În fig. 8.8 (b) este prezentată imaginea wireframe a unui petic Bézier al cărui poliedru de control este reprezentat în fig. 8.8 (a).

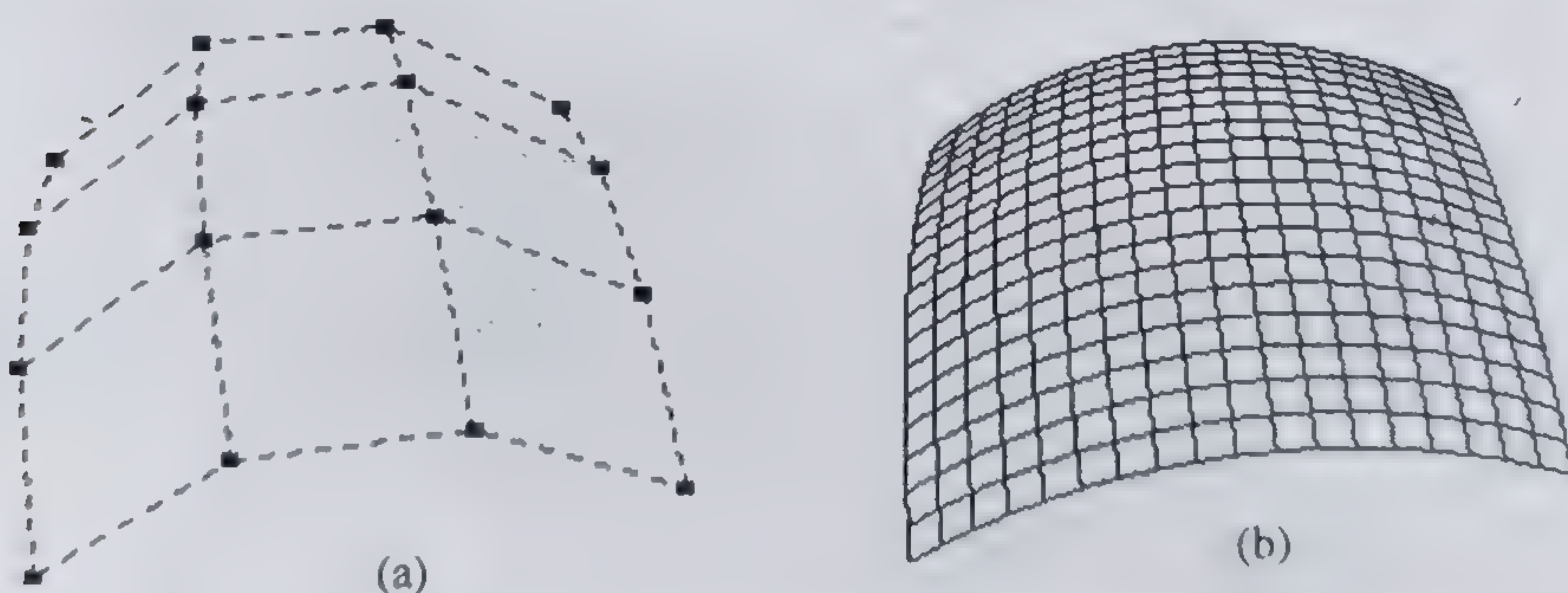


Fig. 8.8 (a) Poliedrul de control.  
(b) Petic de suprafață bicubică Bézier rezultat.

Proprietățile curbelor Bézier se extind în domeniul suprafețelor. Figura 8.9 arată cum se deformează un petic atunci când unul din vârfurile poliedrului de control este "tras" în sus.



Modelul unui petic Bézier se reprezintă prin lista punctelor lui de control în sistemul de referință local (de modelare). În forma cea mai simplă, memorat ca un tablou de  $4 \times 4 \times 3$  numere reale, modelul peticului din fig. 8.8 este:

```
double points[4][4][3] = {
    {{-3.0, -1.0, 4.0},
     {-1.0, 1.0, 4.0},
     { 1.0, 1.0, 4.0},
     { 3.0, -1.0, 4.0}},
    {{-3.0, -0.0, 2.0},
     {-1.0, 2.0, 2.0},
     { 1.0, 2.0, 2.0},
     { 3.0, -0.0, 2.0}},
    {{-3.0, -0.0, 0.0},
     {-1.0, 2.0, 0.0},
     { 1.0, 2.0, 0.0},
     { 3.0, -0.0, 0.0}},
    {{-3.0, -1.0, -2.0},
     {-1.0, 1.0, -2.0}, // se modifica la y = 3
     { 1.0, 1.0, -2.0},
     { 3.0, -1.0, -2.0}},
};
```

Modificarea coordonatei  $y$  a celui de-al doilea punct din ultimul poligon de control (marcată în comentariu) produce modificarea prezentată în fig. 8.9.

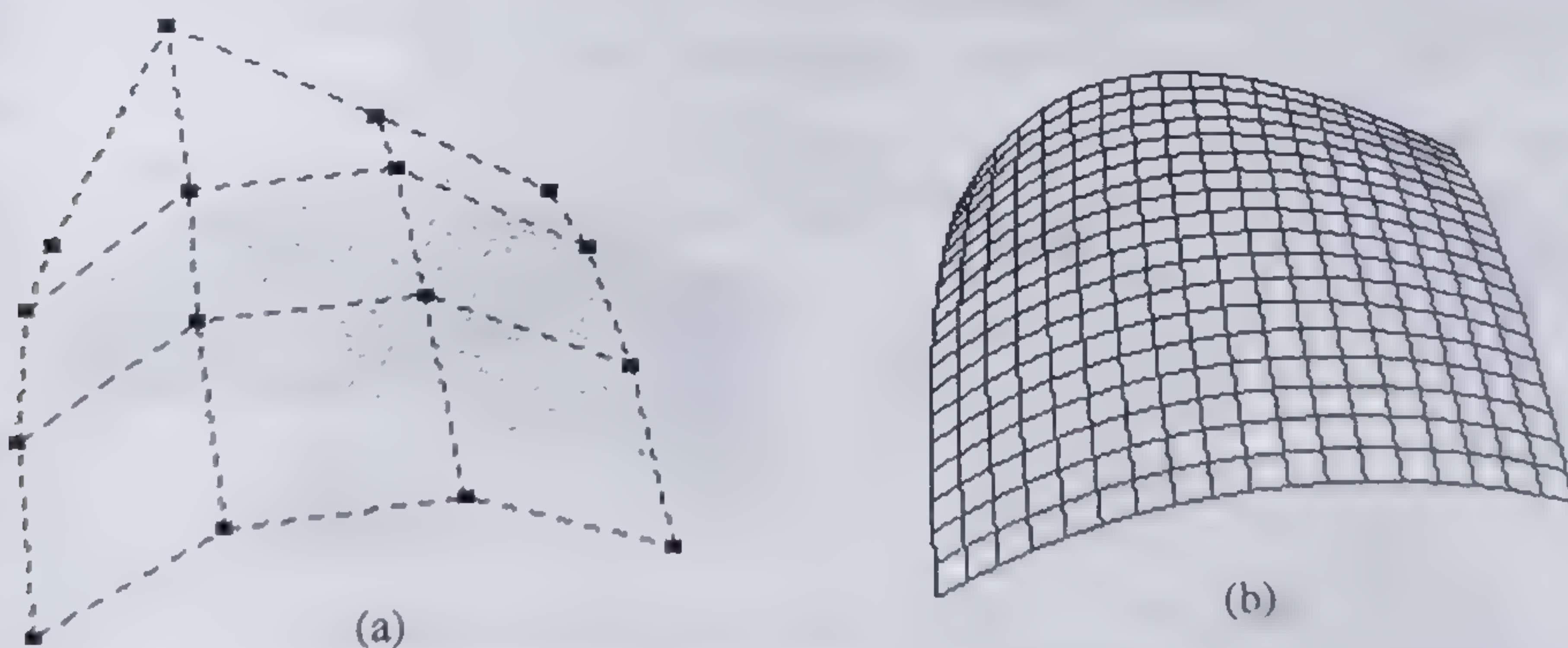


Fig. 8.9 Efectul "ridicării" unui punct de control al poliedrului caracteristic al unui petic Bézier.

Suprafețe mai complexe se obțin prin combinarea peticelor Bézier, cu condiția menținerii continuității între acestea. Pentru asigurarea continuității pozitionale (de ordin 0) între două petice  $Q(u, v)$  și  $R(u, v)$  trebuie să fie îndeplinită una din condițiile:

$$Q(1, v) = R(0, v) \text{ sau } Q(0, v) = R(1, v), \text{ pentru } 0 \leq v \leq 1 \text{ sau}$$

$$Q(u, 1) = R(u, 0) \text{ sau } Q(u, 0) = R(u, 1), \text{ pentru } 0 \leq u \leq 1.$$



Această condiție înseamnă cele două petice au patru puncte de control comune (deci poliedrele caracteristice ale celor două petice au trei muchii comune). În fig. 8.10 este prezentată imaginea a două petice Bézier alipite cu continuitate pozițională. Se poate observa curbura accentuată care apare la această alipire.

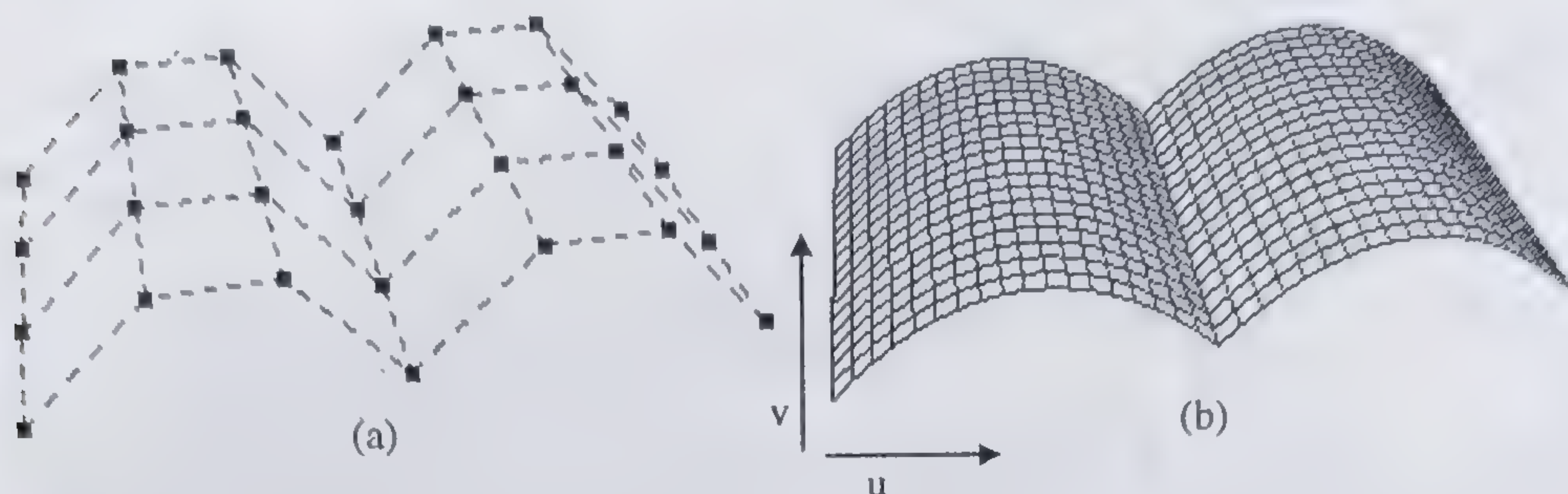


Fig. 8.10 Continuitatea pozițională între două petice Bézier:  
(a) poliedrele caracteristice; (b) peticele Bézier.

Pentru satisfacerea *continuității tangențiale* (de ordinul 1) trebuie ca vectorul tangent la primul petic la  $u = 1$  să aibă aceeași valoare cu vectorul tangent la cel de-al doilea petic la  $u = 0$ , pentru toate valorile lui  $v$  cuprinse în intervalul  $[0,1]$ , dacă alipirea se face de-a lungul muchiei cu  $u = 1$  a primului petic. Condiții similare se pot deduce dacă alipirea are loc pe alte muchii ale peticelor. În fig. 8.11 este arătată alipirea a două petice cu continuitate tangențială.

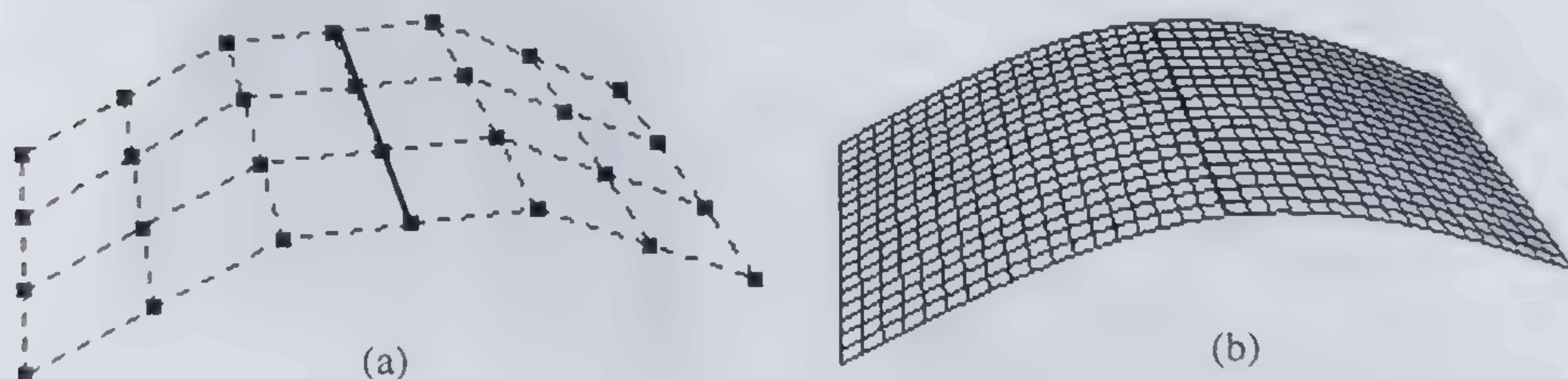


Fig. 8.11 Continuitatea tangențială a două petice Bézier:  
(a) poliedrele caracteristice; (b) peticele Bézier.

Condiția de continuitate tangențială este foarte restrictivă, deoarece necesită ca 4 grupe de câte 3 puncte de control să fie coliniare. La adăugarea unui petic nou, 4 puncte de control sunt deja fixate (punctele de alipire), iar alte 4 trebuie să se afle pe direcții impuse de peticul precedent pentru asigurarea coliniarității respective. Deci rămân puține puncte de control a formei peticului, ceea ce reprezintă o dificultate deosebită de proiectare. O condiție ceva mai puțin restrictivă, dar acceptabilă de alipire a două petice, dezvoltată de Bézier în 1972, este continuitatea tangentelor pe frontiera de adiacență.



Mai trebuie menționat faptul că nu orice formă poate fi aproximată prin rețele de petice patrulatere, precum cele descrise anterior. De exemplu, la modelarea unei sfere (sau elipsoid), la poli peticele patrulatere degenerază în petice triunghiulare.

## 8.4 SUPRAFETE B-SPLINE

Un petic bicubic B-spline este definit de ecuația:

$$Q(u, v) = \sum_{i=0}^n \sum_{j=0}^m P_{ij} B_{ij}(u, v) \quad (8.9)$$

unde  $P_{ij}$  este un tablou de puncte de control, iar  $B_{ij}(u, v)$  este o funcție de bază de două variabile  $u$  și  $v$ , care se poate genera prin produsul funcțiilor de bază ale curbilor B-spline:

$$B_{ij}(u, v) = B_i(u) B_j(v)$$

Peticele B-spline se consideră rectangulare, definite prin poliedrul caracteristic și printr-un tablou de noduri care formează o grilă în planul parametrilor  $u$  și  $v$ .

Analog definirii curbilor, un petic de suprafață B-spline poate fi compus din mai multe segmente de petice, fiecare segment de petic necesitând o grilă de  $4 \times 4$  puncte de control care sunt combinate cu  $4 \times 4$  funcții de bază, definite peste un tablou de  $8 \times 8$  valori ale nodurilor. În fig. 8.12 este reprezentat cu linie punctată un poliedru caracteristic format din  $4 \times 4$  puncte. În primul caz (a) suprafața este un petic B-spline uniform, cu nodurile distribuite regulat în planul parametrilor  $u$  și pe  $v$  în pozițiile  $[0, 1, 2, 3, 4, 5, 6, 7]$  pentru fiecare din parametri. În cel de-al doilea caz (b), suprafața este un petic B-spline neuniform, cu nodurile distribuite neuniform pe  $u$  și pe  $v$  în pozițiile  $[0, 0, 0, 0, 1, 1, 1, 1]$ .

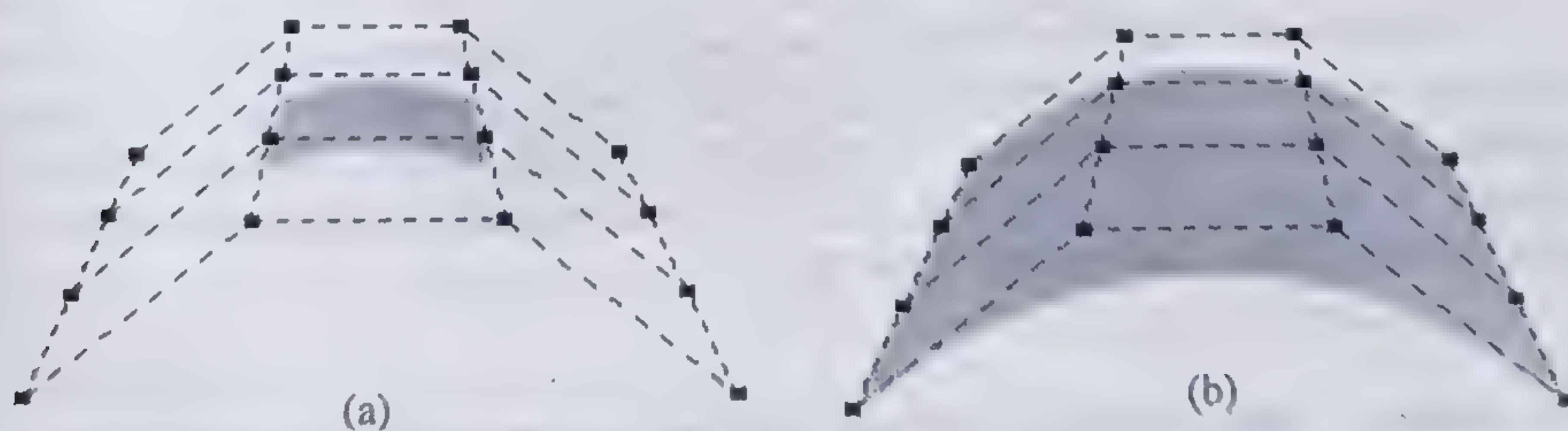


Fig. 8.12 Suprafețe B-spline: (a) petic B-spline uniform; (b) petic B-spline neuniform.

La fel ca și în cazul curbilor B-spline, o suprafață B-spline uniformă nu interpolează punctele marginale ale poliedrului caracteristic (fig. 8.12 (a)). Multiplicitatea nodurilor în cazul suprafețelor B-spline neuniforme asigură interpolarea punctelor marginale ale poliedrului caracteristic (fig. 8.12(b)).



Suprafețe complexe se obțin din mai multe segmente de petice alăturate, prin extinderea poliedrului caracteristic și a tabloului nodurilor.

## 8.5 EXTINDEREA CONTROLULUI PARAMETRIC: NURBS ȘI $\beta$ -SPLINE

Două importante extinderi ale curbelor și suprafețelor B-spline permit controlul formei atât prin poziția punctelor de control, cât și prin alți parametri.

**Curbele și suprafețele B-spline neuniforme raționale** (NURBS – *Non-Uniform Rational B-Spline*) extind curbele și suprafețele B-spline neuniforme prin adăugarea unui parametru suplimentar fiecărui punct de control. Fiecare punct de control se reprezintă într-un sistem cu 4 coordonate, coordonata suplimentară  $w$  fiind un parametru care ponderează efectul fiecărui punct de control.

O curbă NURBS este definită prin puncte de control reprezentate cu patru dimensiuni:

$$P_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$$

Proiecția perspectivă a unei astfel de curbe în spațiul tridimensional se numește curbă B-spline rațională și este definită prin ecuația:

$$Q(u) = \sum_{i=0}^n P_i R_i(u), \text{ unde } R_i(u) = \frac{B_i(u)w_i}{\sum_{j=0}^n B_j(u)w_j} \quad (8.10)$$

Coordonata  $w$  este coordonata omogenă și este folosită pentru diferite transformări geometrice ale punctelor, poligonului caracteristic.

Curbele și suprafețele B-spline raționale au aceleași proprietăți ca și curbele B-spline neuniforme neraționale dacă  $w = 1$ .

**Controlul  $\beta$ -spline** adaugă doi parametri globali care afectează întreaga curbă sau suprafață; în forma cea mai simplă acești parametri sunt tensiunea (*tension*) și baza (*bias*). Formele  $\beta$ -spline pot fi considerate o generalizare a formelor B-spline și sunt utilizate în proiectarea asistată de calculator pentru obținerea unor forme tridimensionale complexe.

## 8.6 REDAREA SUPRAFETELOR PARAMETRICE

Redarea suprafețelor parametrice se poate face prin două metode:

- Aproximarea suprafeței parametrice printr-o rețea de poligoane planare și redarea acestora folosind procedurile "standard" de redare.
- Redarea directă, din descrierea parametrică a suprafețelor.



În mod obișnuit, prima modalitate, prin aproximare poligonală, este cea mai eficientă și este cel mai frecvent folosită.

Se pune întrebarea justificată, de ce să fie folosită reprezentarea parametrică care modelează un obiect cu o precizie ridicată, ca apoi redarea să fie efectuată prin aproximare (mai puțin precisă) cu fețe poligonale. Există două motive pentru care se procedează în acest mod. Primul motiv este acela că în sistemele de proiectare interactive reprezentarea parametrică permite flexibilitate și simplitate în definirea formelor obiectelor, calități care nu se obțin ușor dacă proiectantul este obligat să introducă fiecare poligon separat. Cel de-al doilea motiv este că, de la un model precis reprezentat parametric, se poate trece la aproximarea poligonală cu controlul complet asupra dimensiunii poligoanelor și deci a preciziei de aproximare. Acest lucru permite, pe de o parte, modelarea adaptivă, prin introducerea unui număr mai mare de poligoane în regiunile cu curbura accentuată a obiectului și, pe de altă parte, obținerea unui model cu nivele multiple de detaliu pornind de la o singură reprezentare, reprezentarea parametrică a obiectului.

Redarea directă a suprafețelor parametrice este dificilă deoarece nu este posibilă aplicarea tehnicilor de calcul eficient pentru conversia unui petic de suprafață din descrierea parametrică în mulțimea de pixeli care corespund imaginii acestora. Pentru poligoane planare, operația de conversie (care a fost prezentată în capitolul 5) se implementează eficient, prin interpolare atât pentru coordonatele în planul de afișare, cât și pentru adâncimi, umbrire etc. Pentru suprafețele parametrice nu se pot aplica astfel de tehnici și redarea necesită calcule complexe pentru fiecare element al suprafeței parametrice, ceea ce reprezintă un consum de timp de calcul important. De aceea, redarea directă a suprafețelor parametrice este rareori folosită în aplicațiile grafice interactive sau de timp real, fiind abordată în aplicații de creare a unor imagini statice foarte precise a obiectelor.

Aproximarea unei suprafețe parametrice cu o rețea de poligoane planare se obține folosind curbe izoparametrice (cu valoare constantă a unui parametru). Prin intersecția curbelor izoparametrice se generează o rețea de puncte ale suprafeței care se folosesc ca vârfuri ale rețelei de poligoane planare. Imaginile din figurile acestui subcapitol au fost obținute prin redarea wireframe sau plină a rețelei de aproximare poligonală a suprafețelor parametrice.

Un aspect important în generarea rețelei poligonale este stabilirea rezoluției parametrice de aproximare. O subdiviziune prea fină, folosind un număr mare de curbe izoparametrice, la intervale mici ale parametrului respectiv, produce un număr mare de poligoane și deci implică cerințe mari de timp redare a acestora. O subdiviziune grosieră, folosind un număr mic de curbe isoplanare produce discontinuități vizibile la redarea poligoanelor rezultate.

O soluție mai bună este divizarea cu rezoluție variabilă a suprafețelor: zonele de suprafață cu o curbura accentuată sunt divizate pe intervale mai mici, iar zonele cu curbura locală mică sunt divizate pe intervale mai mari. Problemele care apar în cazul divizării cu rezoluție variabilă (posibilitatea apariției "găurilor") sunt soluționate prin înlocuirea evaluării punctelor pe curbele izoparametrice cu subdivizarea acestora prin introducerea unor puncte de control suplimentare [Watt95].



## 8.7 FUNCȚII OPENGL PENTRU REDAREA SUPRAFETELOR PARAMETRICE

În OpenGL sunt prevăzute posibilități de generare și redare a curbelor și suprafețelor Bézier folosind funcții numite evaluatori, care calculează coordonatele punctelor de pe o curbă sau o suprafață Bézier, în funcție de valorile parametrilor ( $u$  sau  $u$  și  $v$ ). Aceste puncte sunt grupate în primitive geometrice, care sunt redare pe display. Funcțiile de evaluare a coordonatelor sunt `glEvalCoord1#()` și `glEvalCoord2#()` pentru curbe, respectiv suprafețe Bézier.

### 8.7.1 GENERAREA ȘI REDAREA CURBELOR BÉZIER

Programul OpenGL prin care s-a obținut prima curbă Bézier din fig. 8.1 este dat în exemplul următor.

#### ■ Exemplul 8.1

```
#include <GL/glut.h>
#define M 20
GLfloat points[4][3] = {
    { -1.0, -1.0, 0.0 },
    { -1.0, 1.0, 0.0 },
    { 1.0, 1.0, 0.0 },
    { 1.0, -1.0, 0.0 }
};

void Init(void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glPointSize(5);
    glLineStipple(1, 0x0F0F);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_FLAT);
}

void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_MAP1_VERTEX_3);
    glColor3f(0.0, 0.0, 0.0);
    glPushMatrix();
    glTranslated(0, 0, -7);
    glDisable(GL_LINE_STIPPLE);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,
            &points[0][0]);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i <= M; i++)
        glEvalCoord1f((GLfloat)i/M);
    glEnd();
    // Afisare puncte de control
    glPointSize(5);
    glBegin(GL_POINTS);
```



```

    for (i = 0; i < 4; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    // Afisare poligon caracteristic
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 4; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 0.1, 1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
                        GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Bezier");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```

Programul afișează un segment de curbă Bézier cu linie plină, punctele de control ca mici pătrate și poligonul caracteristic cu linie punctată (prin validarea `glEnable(GL_LINE_STIPPLE)` și setarea succesiunii de linii întrerupte cu funcția `glLineStipple()`).

Cele patru punctele de control ale unei curbe Bézier sunt memorate în tabloul `points[4][3]` care este unul din argumentele funcției `glMap1f()` care definește tipul de corespondență între datele de intrare și datele calculate de funcția de evaluare `glEvalCoord1#()`. Prototipurile funcției `glMap1#()` sunt următoarele:

```

void glMap1d(GLenum target, GLdouble u1, GLdouble u2,
             GLint stride, GLint order, const GLdouble *points);
void glMap1f(GLenum target, GLdouble u1, GLdouble u2,
             GLint stride, GLint order, const GLfloat *points);

```



Argumentul `target` specifică tipul de puncte de control prevăzute în vectorul `points` și ce tip de date de ieșire va genera funcția de evaluare `glEvalCoords()` care va fi apelată ulterior. Acest argument poate lua mai multe valori de constante simbolice, specificând coordonate de vârfuri (`GL_MAP1_VERTEX3`), normale (`GL_MAP1_NORMAL`) sau coordonate de textură (`GL_MAP_TEXTURE_COORD_1, ..., GL_MAP_TEXTURE_COORD_4`). Intervalul `u1 - u2` este intervalul care se mapează liniar în intervalul  $[0,1]$  al parametrului `u` pentru calculul expresiei polinomiale  $Q(u')$  (relația 8.3), unde  $u' = (u - u1)/(u2 - u1)$ :

$$Q(u') = \sum_{i=0}^n P_i B_{i,n}(u')$$

$B_{i,n}(u)$  sunt polinoamele lui Bernstein de gradul  $n$ , iar  $P_i$  sunt punctele de control memorate în vectorul `points`.

Argumentul `stride` specifică numărul de valori în virgulă flotantă cu care se avansează în vectorul `points` de la un punct de control la următorul (în exemplul de mai sus are valoarea 3). Argumentul `order` reprezintă ordinul curbei, și este egal cu gradul polinomului Bernstein plus 1. Vectorul `points` conține punctele de control care sunt folosite de funcția de evaluare. În felul acesta sunt definite toate datele necesare evaluării relației (8.3): gradul polinomului Bernstein, punctele de control și intervalul de mapare al parametrului `u`.

Pentru validarea unui anumit tip de evaluare se apelează mai întâi funcția `glEnable()` cu argument corespunzător argumentului `target` al funcției `glMap1#()` (deci poate fi `GL_MAP1_VERTEX3`, etc.).

Funcția `glEvalCoord1f(GLfloat u)` calculează coordonatele unui punct pe curbă pentru valoarea dată a parametrului `u` și lansează comenzi OpenGL corespunzătoare tipului punctelor de control. De exemplu, dacă punctele de control sunt coordonate de vârfuri (`GL_MAP1_VERTEX3`), atunci, pentru fiecare valoare a parametrului `u`, coordonatele calculate sunt folosite ca argumente ale unei funcții `glVertex#()`.

Forma și dimensiunea segmentului de curbă Bézier se poate modifica prin modificarea poziției punctelor de control. Familia de curbe din fig. 8.1(a) s-a obținut prin modificarea poziției punctului de control  $P_1$ . La fel, imaginile din fig. 8.13 se obțin dacă se rulează același program, dar cu vectorul de puncte modificat astfel:

```
GLfloat points[4][3] = {
    { -1.0, -1.0, 0.0 },
    {  0.0,  1.0, 0.0 },
    {  1.0, -1.0, 0.0 },
    {  2.0,  1.0, 0.0 } };
```

Curba este aproximată printr-un număr de  $M$  segmente de dreaptă,  $M$  fiind stabilit în programul de aplicație în funcție de precizia de redare dorită. Dacă numărul de segmente de aproximare este redus, se observă eroarea de aproximare a curbei (fig. 8.13 (b)).



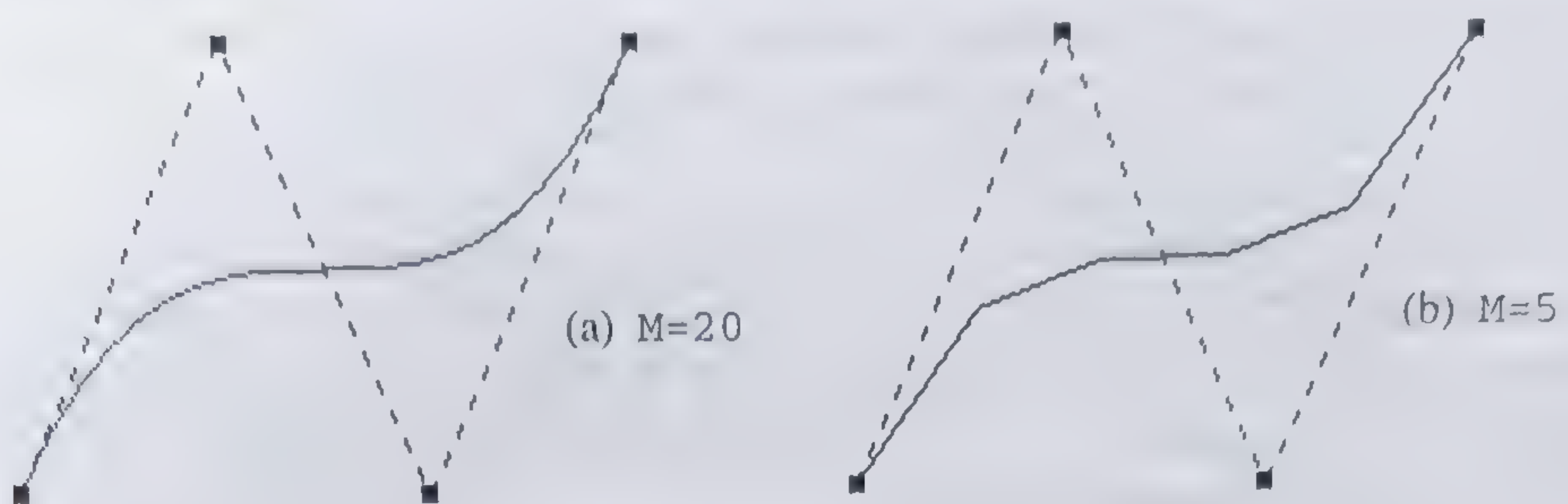


Fig. 8.13 Curbă Bézier cubică, punctele de control și poligonul caracteristic.

### 8.7.2 GENERAREA ȘI REDAREA SUPRAFETELOR BÉZIER

Pentru redarea suprafețelor Bézier se apelează o funcție de evaluare bidimensională `glEvalCoord2#()` care calculează un punct de pe suprafață în funcție de doi parametri  $u$  și  $v$ . În exemplul următor se poate urmări modul de redare a unui petec de suprafață bicubică Bézier.

#### ■ Exemplul 8.2

```
#include <GL/glut.h>
#define M 20
GLfloat points[4][4][3] = {
    {{-1.5, -1.5, 4.0},
     {-0.5, -1.5, 2.0},
     { 0.5, -1.5, -1.0},
     { 1.5, -1.5, 2.0}},
    {{-1.5, -0.5, 1.0},
     {-0.5, -0.5, 3.0},
     { 0.5, -0.5, 0.0},
     { 1.5, -0.5, -1.0}},
    {{-1.5, 0.5, 4.0},
     {-0.5, 0.5, 0.0},
     { 0.5, 0.5, 3.0},
     { 1.5, 0.5, 4.0}},
    {{-1.5, 1.5, -2.0},
     {-0.5, 1.5, -2.0},
     { 0.5, 1.5, 0.0},
     { 1.5, 1.5, -1.0}}
};

void Display() {
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
            0, 1, 12, 4, &points[0][0][0]);
}
```



```

glEnable(GL_MAP2_VERTEX_3);
glMapGrid2f(M, 0.0, 1.0, M, 0.0, 1.0);
glColor3f(0.0, 0.0, 0.0);
glPushMatrix ();
glTranslated(0.0,0.0,-10.0);
glRotatef(20.0, 0.0, 1.0, 0.0);
glRotatef(-10.0, 0.0, 0.0, 1.0);
glEvalMesh2(GL_FILL, 0, M, 0, M);
glPopMatrix ();
glutSwapBuffers();
}

```

Toate celelalte funcții ale programului sunt identice cu cele din programul precedent și nu au mai fost prezentate. La execuția programului cu parametrul  $M=20$  se obține reprezentarea wireframe a suprafeței Bézier dată prin cele 16 puncte de control memorate în vectorul `points` (fig. 8.14(a)).

Funcția `glEvalCoord2f(GLfloat u, GLfloat v)` evaluează un punct pe suprafață pentru valorile  $u$  și  $v$  ale parametrilor curbei și folosește valorile calculate ca argumente ale funcțiilor apelate conform tipului de corespondență definit de funcția `glMap2#()`. Funcția `glMap2#()` stabilește modul de evaluare a punctelor de pe o suprafață Bézier pentru cei doi parametri. Argumentele de apel sunt similare celor folosite pentru funcția `glMap1#()`, cu deosebirea că se definește ordinul curbei și intervalul de calcul pentru fiecare parametru.

Suprafața curbă este aproximată cu o rețea de poligoane planare și redată de funcția `glEvalMesh2()` care specifică modul de redare al poligoanelor (care poate fi `GL_FILL`, `GL_POINT`, `GL_LINE`) și intervalele de eșantionare a suprafeței pentru parametrii  $u$  și  $v$ . Dimensiunile acestor intervale sunt definite prin funcția `glMapGrid2f()`.

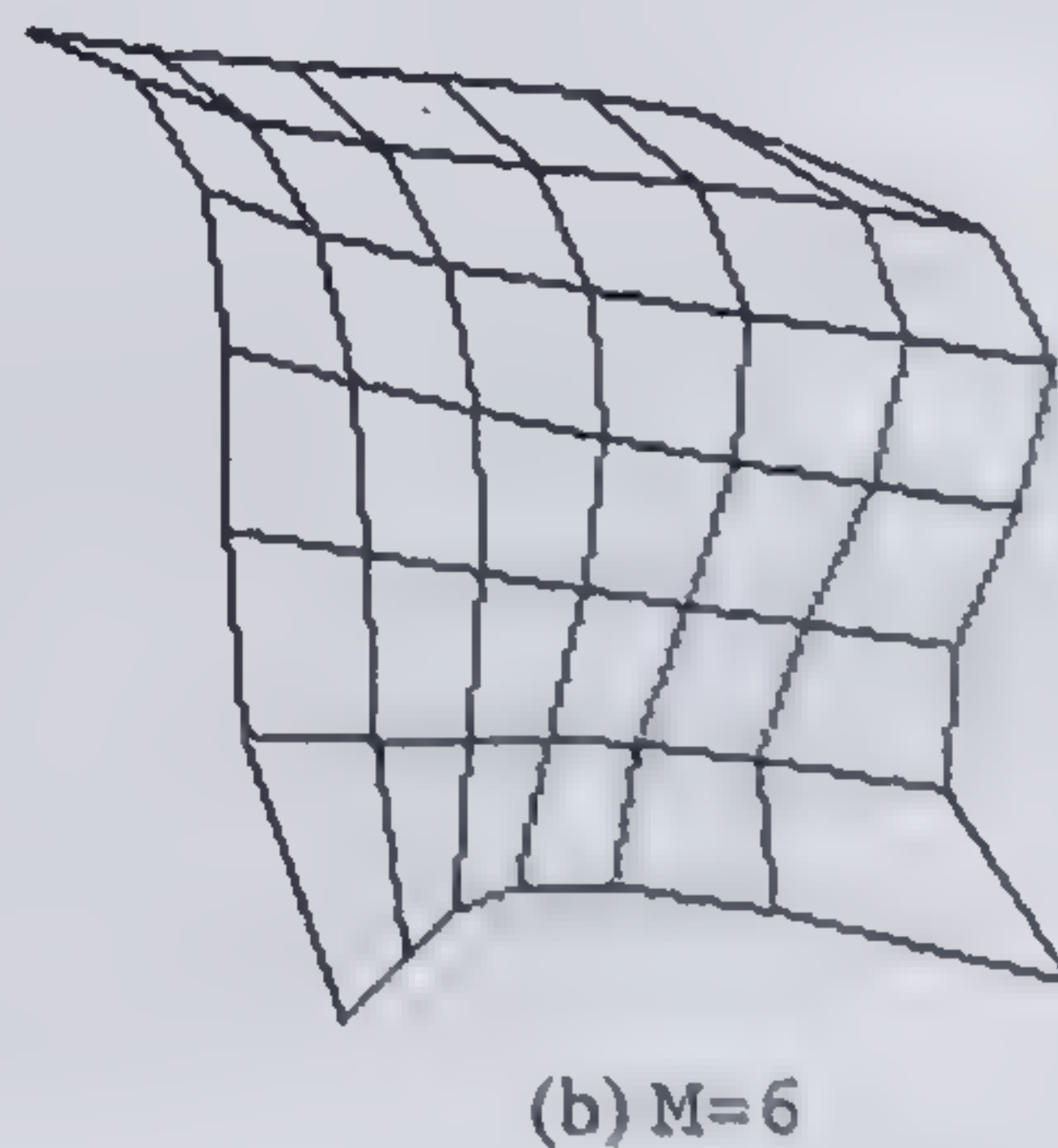
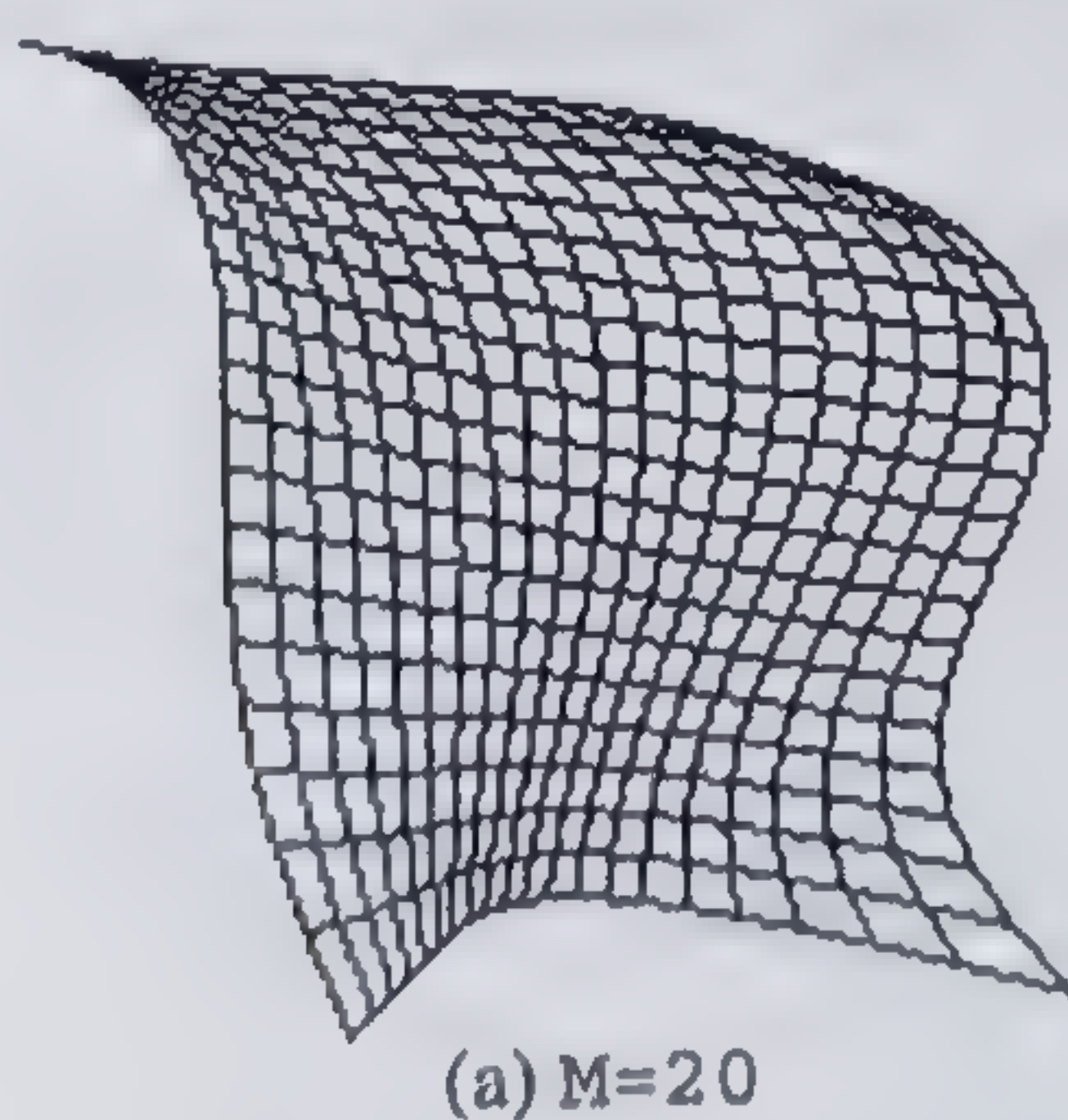


Fig. 8.14 Suprafață Bézier reprezentată wireframe cu diferite precizii.

În Planșa 4 este redată aceeași suprafață Bézier cu iluminare și umbrire.



### 8.7.3 GENERAREA ȘI REDAREA CURBELOR ȘI A SUPRAFETELOR B-SPLINE

Biblioteca OpenGL împreună cu utilitarul GLU, permite generarea și redarea curbelor și a suprafețelor B-spline în forma cea mai generală, și anume B-spline neuniforme raționale (NURBS - *Non-Uniform Rational B-Spline*). Prin alegerea parametrului de ponderare  $w = 1$  ( $w$  fiind a patra coordonată în spațiul omogen tridimensional) pentru toate punctele de control, formele B-spline raționale au aceleași proprietăți cu formele B-spline neraționale. Din acestea se pot defini forme B-spline uniforme sau neuniforme în funcție de intervalul parametric al nodurilor.

Funcțiile de evaluare (evaluatori) `glEvalCoord#()` sunt primitive OpenGL care permit evaluarea punctelor de pe curbe și suprafețe parametrice depinzând de modul de definire a evaluării. Evaluatorii pot fi implementați eficient în hardware. Pentru reprezentarea curbelor și a suprafețelor B-spline neuniforme raționale sunt necesare funcții mai complexe care utilizează evaluatori OpenGL. Aceste funcții sunt prevăzute de biblioteca auxiliată GLU.

O curbă sau o suprafață NURBS se definește ca un obiect de tipul `GLUnurbsObj` care se alocă în memoria liberă (heap). Pentru un astfel de obiect se specifică parametri de definiție: punctele de control și vectorul de noduri (pentru unul sau doi parametri). Pentru redarea imaginii unei curbe sau suprafețe NURBS se mai specifică datele de eșantionare, care definesc intervalele la care se evaluează punctele obiectului și aceste puncte sunt grupate în primitive geometrice (linii sau poligoane). În exemplele următoare este descris pe scurt modul de generare și redare a curbelor și suprafețelor NURBS.

#### ■ Exemplul 8.3

Programul care urmează generează și redă o curbă NURBS folosind funcțiile bibliotecii GLU.

```
#include <GL/glut.h>
GLUnurbsObj *theNurb;
GLfloat points[9][3] = {
    { -1.0,  4.0,  0.0},
    { -2.5,  3.0,  0.0},
    { -3.0, -0.5,  0.0},
    { -1.5, -4.0,  0.0},
    {  0.0, -4.0,  0.0},
    {  1.5, -3.0,  0.0},
    {  2.5, -1.5,  0.0},
    {  2.0,  1.0,  0.0},
    {  3.5,  2.0,  0.0}};
void Init(){
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
```



```

glLineStipple(1, 0x0F0F);
glPointSize(5);
theNurb = gluNewNurbsRenderer();
}
void Display() {
    GLfloat knots[13] = {0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6, 6};
    gluNurbsProperty(theNurb,
                     GLU_SAMPLING_TOLERANCE, 1.0);
    int nknots = sizeof(knots)/sizeof(GLfloat);
    int npoints = sizeof(points)/sizeof(GLfloat);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glPushMatrix();
    glTranslatef(0.0, 0.0, -20.0);
    glDisable(GL_LINE_STIPPLE);
    gluBeginCurve(theNurb);
    gluNurbsCurve(theNurb, // pointer obiect NURBS
                  nknots, knots, // noduri
                  3, // interval
                  &points[0][0], // vector puncte control
                  4, // ordinul curbei
                  GL_MAP1_VERTEX_3); // tip evaluator
    gluEndCurve(theNurb);
    // Afisare puncte de control
    glBegin(GL_POINTS);
    for (int i = 0; i < 9; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    // Afisare poligon caracteristic
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 9; i++)
        glVertex3fv(&points[i][0]);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

```

Funcțiile `main()` și `Reshape()` ale programului sunt identice cu cele din programele precedente. Pentru valorile punctelor de control `points` și ale nodurilor `knots` de mai înainte se obține curba din fig. 8.15(a) compusă din șase segmente NURBS, corespunzătoare unui număr de 9 puncte de control și 13 noduri, dintre care nodurile de la capete sunt noduri cu multiplicitatea 4, pentru interpolarea punctelor de la capete.

Obiectul de tip `GLUnurbsObj` este creat dinamic în memoria liberă prin apelul funcției `gluNewNurbsRenderer()` care returnează pointerul la obiectul creat. Acestui obiect i se atribuie mai multe proprietăți de redare, dintre care s-a specificat eroarea de eșantionare a curbei la redare prin funcția `gluNurbsProperty()`.



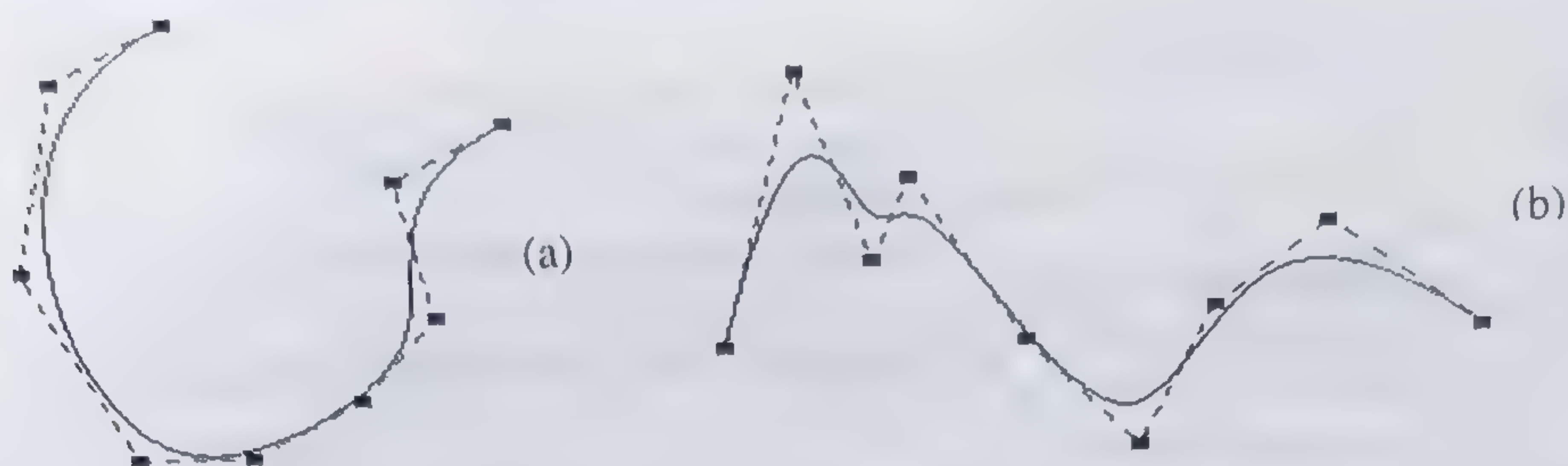


Fig. 8.15 Curbe NURBS

Vectorul punctelor de control (*points*), vectorul nodurilor (*knots*) și tipul funcției de evaluare (*GL\_MAP1\_VERTEX\_3*) sunt parametri care se transmit prin funcția *gluNurbsCurve()* care generează sau redă o curbă NURBS. Fiecare curbă se redă sau se generează într-un bloc de program cuprins între instrucțiunile *gluBeginCurve()* și *gluEndCurve()*. Afișarea punctelor de control și a poligonului caracteristic se face la fel ca în programul precedent. Dacă se execută același program, cu același vector de noduri, dar cu un alt vector de puncte de control, se obține curba din fig. 8.15(b).

#### ■ Exemplul 8.4

Pentru generarea unei suprafețe NURBS se creează tot un obiect de tipul *GLUnurbsObj*, dar se definește un poliedru caracteristic (printr-un tablou de puncte de control) și doi vectori (care pot fi identici) de noduri, pentru cei doi parametri *u* și *v* ai suprafeței. Programul de generare a unei suprafețe NURBS este următorul:

```
#include <GL/glut.h>
GLfloat ctlpoints[4][4][3];
GLUnurbsObj *theNurb;
void init_points(void) {
    int u, v;
    for (u = 0; u < 4; u++) {
        for (v = 0; v < 4; v++) {
            ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
            ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);
            if ((u == 1 || u == 2) && (v == 1 || v == 2))
                ctlpoints[u][v][2] = 3.0;
            else ctlpoints[u][v][2] = -3.0;
        }
    }
}
void Init() {
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    init_points();
    theNurb = gluNewNurbsRenderer();
}
```



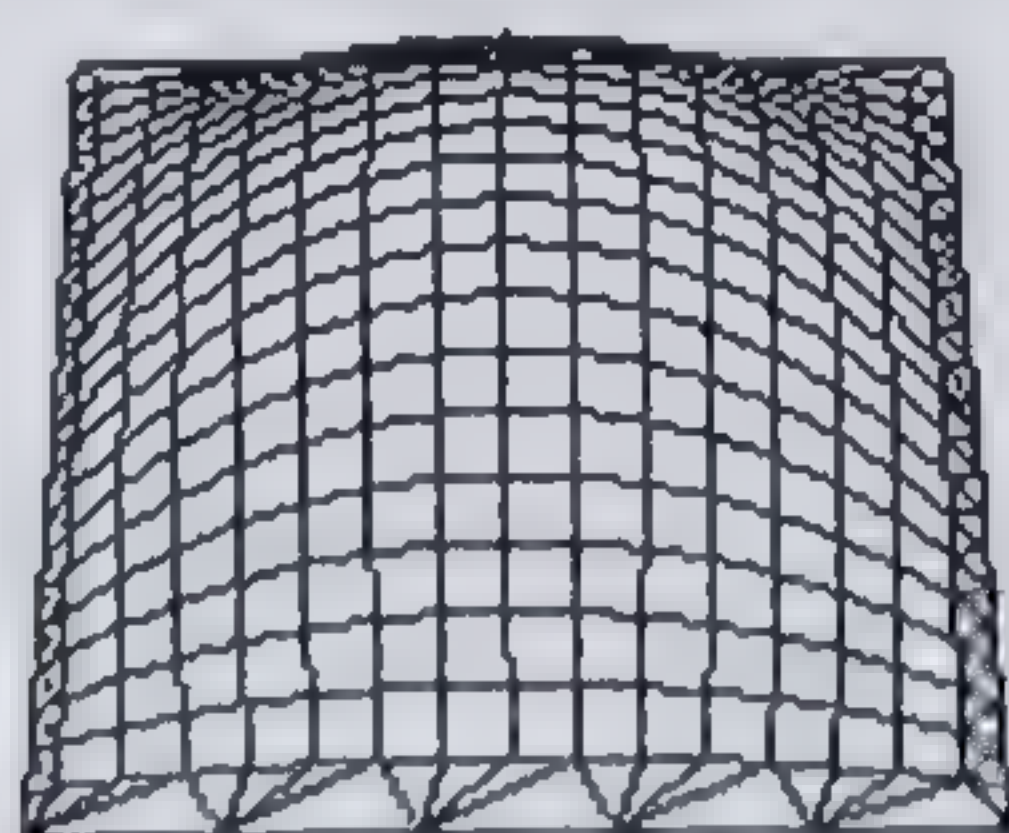
```

void Display(){
    gluNurbsProperty(theNurb, GLU_DISPLAY_MODE,
                     GLU_OUTLINE_POLYGON);
    gluNurbsProperty(theNurb,
                     GLU_SAMPLING_TOLERANCE, 30);
    GLfloat knots[8] = {0,0,0,0,1,1,1,1};
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    glPushMatrix();
    glTranslatef(0.0,0.0,-20.0);
    glRotatef(330.0, 1.,0.,0.);
    gluBeginSurface(theNurb);
    gluNurbsSurface(theNurb,    // pointer obiect NURBS
                    8, knots, 8, knots,    // vector noduri pe u, v
                    4*3, 3,    // intervale pe u, v
                    &ctlpoints[0][0][0], // tablou puncte control
                    4, 4,    // ordin u, v
                    GL_MAP2_VERTEX_3);    // tip evaluator
    glEndSurface(theNurb);
    glPopMatrix();
    glutSwapBuffers();
}

```

Imaginea creată la execuția acestui program este dată în fig. 8.16(a). Generarea și redarea suprafeței este realizată de funcția `gluNurbsSurface()`, căreia i se transmit ca argumente tabloul de puncte de control (`ctlpoints`), vectorii de noduri (`knots`) și tipul evaluatorului (`GL_MAP2_VERTEX_3`). Tabloul de puncte de control este generat de funcția `init_points()`.

Modul de redare a fețelor poligonale prin care se aproximează peticul de suprafață este stabilit prin funcția `gluNurbsProperty()` cu tipul de redare `GLU_OUTLINE_POLYGON` pentru redare wireframe (fig. 8.16 (a)), respectiv `GLU_FILL` pentru redarea plină a fețelor (fig. 8.16 (b)).



(a)



(b)

Fig. 8.16 Suprafață NURBS redată wireframe (a) și cu fețe iluminate (b).

În cele două exemple 8.3 și 8.4 s-au definit forme NURBS cu coordonata de ponderare  $w = 1$ , deoarece în evaluarea `GL_MAP1_VERTEX3` sau `GL_MAP2_VERTEX3` punctele de control sunt date cu trei coordonate ( $x, y, z$ ) și se consideră coordonata  $w = 1$ . Astfel de forme NURBS au aceleași proprietăți ca și formele B-spline neraționale neuniforme.



## ANTI-ALIASING

Se poate observa ușor că liniile afișate pe display apar ca o secvență de segmente care dau un aspect zimțat (în trepte de scară), aspect cunoscut sub numele de *aliasing*. În text se va păstra termenul original de *aliasing*, dat fiind că traduceriile sunt mai puțin sugestive. Cauza fundamentală a *aliasing*-ului în grafica pe calculator este crearea imaginilor prin eșantionare în domeniul spațiului și, pentru imaginile animate, și în domeniul timp. Pentru generarea unei imagini, pasul final îl constituie calculul intensității pixelilor, care reprezintă trecerea de la intensitatea  $i(x,y)$  definită în spațiul bidimensional continuu, la intensitatea fiecărui pixel în spațiul imagine discret. Această trecere este echivalentă cu eșantionarea spațiului bidimensional continuu cu o grilă de puncte de eșantionare considerate, de exemplu, în centrul fiecărui pixel.

Cea mai familiară manifestare a *aliasing*-ului o constituie aspectul de rugozitate (*jagged*) al liniilor sau al muchiilor poligoanelor. De asemenea, suprafețele înguste pot apare întrerupte, în funcție de orientarea lor. Un alt efect al *aliasing*-ului îl constituie apariția și dispariția obiectelor mici care ocupă o suprafață mai mică decât suprafața unui pixel și care sunt redade sau nu în funcție de intersecția cu grila de eșantionare. Aceste efecte sunt mai supărătoare în secvențele de imagini în mișcare. Muchiile în formă de scară sau suprafețele întrerupte dau o senzație de mișcare necontrolată a acestora (*crawl*), care este intolerabilă în multe aplicații de realitate virtuală. Tehnicile de prelucrare a imaginii pentru atenuarea efectelor de *aliasing* sunt cunoscute sub numele de *anti-aliasing*. Aceste tehnici pot fi înțelese și implementate algoritmic pe baza teoriei Fourier.

### 9.1 CONSIDERAȚII TEORETICE ASUPRA ALIASING-ULUI

Imaginea bidimensională în sistemul de referință al porții de afișare este considerată un câmp determinist de dimensiuni infinite  $i(x, y)$ , ale cărei eșantioane se obțin prin multiplicarea cu o funcție de eșantionare spațială  $s(x, y)$ :



$$s(x, y) = \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} \delta(x - n\Delta x, y - m\Delta y) \quad (9.1)$$

Funcția  $s(x, y)$  este compusă dintr-o matrice infinită de funcții delta, aranjate sub forma unei grile (matrice) de distanțe  $(\Delta x, \Delta y)$  (fig. 9.1).

Imaginea eșantionată  $f(x, y)$  este:

$$f(x, y) = i(x, y) s(x, y) = \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} i(n\Delta x, m\Delta y) \delta(x - n\Delta x, y - m\Delta y) \quad (9.2)$$

unde funcția delta bidimensională este  $\delta(n, m) = \begin{cases} 1 & \text{pentru } n = 0, m = 0 \\ 0 & \text{pentru } n \neq 0, m \neq 0 \end{cases}$

Reprezentarea în domeniul frecvenței spațiale  $\omega$  (radiani/distanță) a imaginii eșantionate se obține prin aplicarea transformatei Fourier bidimensionale continue imaginii eșantionate:

$$F(\omega_x, \omega_y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-j(\omega_x x + \omega_y y)} dx dy \quad (9.3)$$

Conform teoremei convoluției, transformata Fourier a imaginii eșantionate poate fi exprimată prin produsul de convoluție dintre spectrul imaginii  $I(\omega_x, \omega_y)$  și spectrul funcției de eșantionare  $S(\omega_x, \omega_y)$ :

$$F(\omega_x, \omega_y) = \frac{1}{4\pi^2} I(\omega_x, \omega_y) \otimes S(\omega_x, \omega_y) \quad (9.4)$$

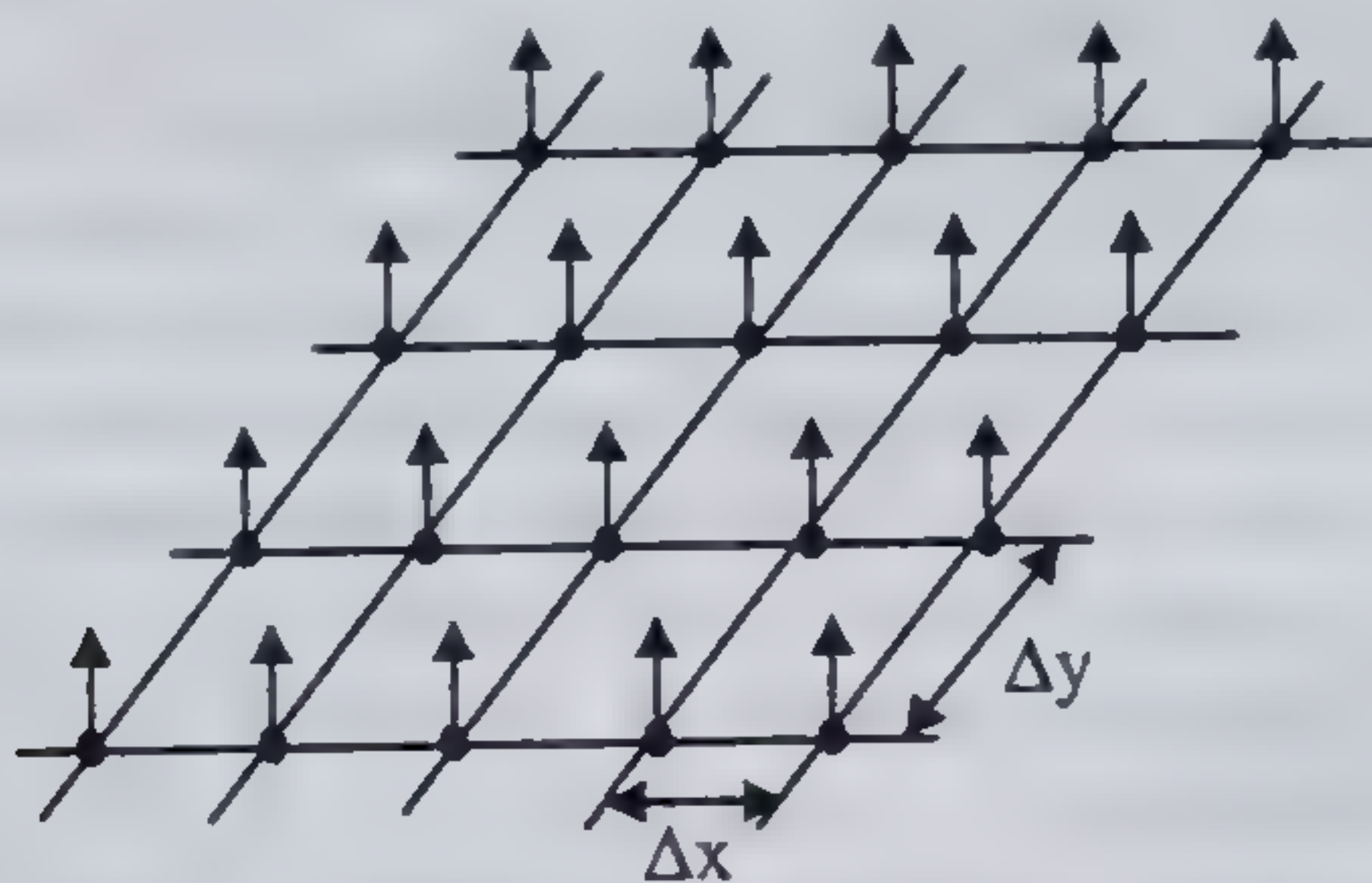


Fig. 9.1 Matricea de eșantionare spațială cu funcții delta.

Transformata Fourier bidimensională a funcției de eșantionare spațială este constituită dintr-o matrice infinită de funcții delta în domeniul frecvenței spațiale:

$$S(\omega_x, \omega_y) = \frac{4\pi^2}{\Delta x \Delta y} \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} \delta(\omega_x - n\omega_{xs}, \omega_y - m\omega_{ys}) \quad (9.5)$$

unde  $\omega_{xs} = \frac{2\pi}{\Delta x}$  și  $\omega_{ys} = \frac{2\pi}{\Delta y}$  reprezintă frecvențele de eșantionare spațială.



Presupunând că imaginea are o bandă limitată (*band-limited*), astfel că:

$$I(\omega_x, \omega_y) = 0 \quad \text{pentru } |\omega_x| > \omega_{xc} \text{ și } |\omega_y| > \omega_{yc} \quad (9.6)$$

și făcând produsul de convoluție cu matricea de eșantionare, se obține:

$$F(\omega_x, \omega_y) = \frac{1}{\Delta x \Delta y} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\omega_x - u, \omega_y - v) \cdot \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} \delta(u - n\omega_{xs}, v - m\omega_{ys}) du dv \quad (9.7)$$

Dacă se schimbă ordinea de însumare și se aplică proprietatea de deplasare a funcției delta, se obține spectrul imaginii eșantionate:

$$F(\omega_x, \omega_y) = \frac{1}{\Delta x \Delta y} \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I(\omega_x - n\omega_{xs}, \omega_y - m\omega_{ys}) \quad (9.8)$$

Reprezentarea grafică a relației (9.8), dată în fig. 9.2(b), evidențiază faptul că spectrul imaginii eșantionate constă din spectrul imaginii originale care se repetă la infinit în planul frecvențelor, pe o matrice de rezoluție  $(2\pi/\Delta x, 2\pi/\Delta y)$ .

Imaginea se reconstruiește printr-un *filtru bidimensional trece-jos*  $H(\omega_x, \omega_y)$  cu frecvențele de tăiere egale cu jumătatea frecvențelor de eșantionare (fig. 9.8(c)).

Se poate considera că frecvențele de eșantionare și frecvențele maxime ale imaginii sunt egale pe cele două coordonate:  $\omega_{xs} = \omega_{ys} = \omega_s$  și  $\omega_{xc} = \omega_{yc} = \omega_c$ . Relația dintre frecvența de eșantionare ( $\omega_s$ ) și frecvența spațială maximă a imaginii (frecvența de tăiere  $\omega_c$ ) exprimă posibilitatea de refacere a imaginii după eșantionare [Hua75].

Dacă frecvența spațială maximă a imaginii este mai mică sau egală cu jumătate din frecvența de eșantionare ( $\omega_c \leq \omega_s/2$ ), atunci, prin filtrarea imaginii eșantionate, se obține imaginea originală nedistorsionată. Intervalul  $\omega_s/2$  este numit *limita Nyquist*.

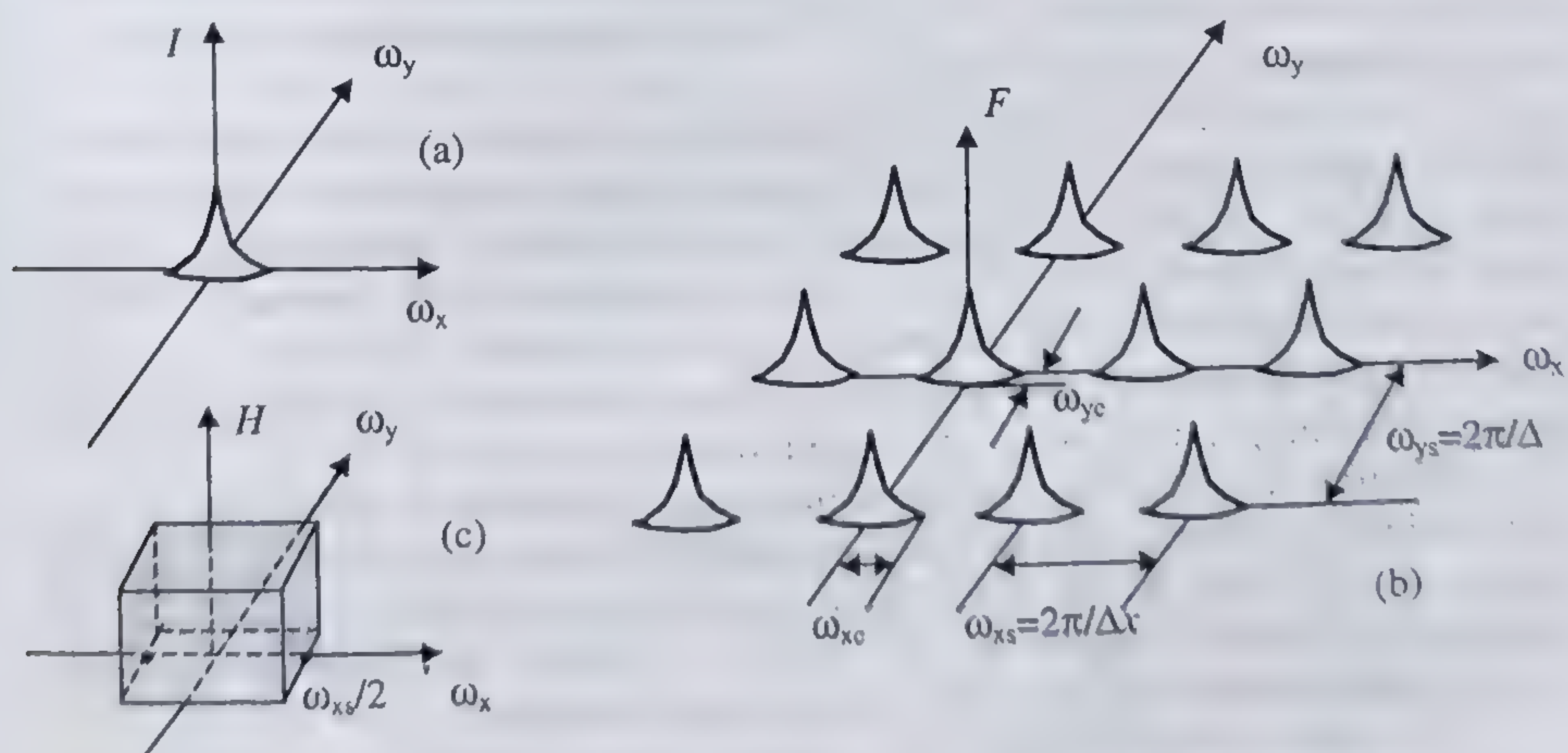


Fig. 9.2 (a) Spectrul imaginii originale. (b) Spectrul imaginii eșantionate. (c) Filtrul de reconstrucție ideal.



Dacă frecvența spațială maximă a imaginii este mai mare decât jumătate din frecvența de eșantionare ( $\omega_c > \omega_s/2$ ), atunci spectrele imaginii eșantionate se suprapun, producând un *zgomot de spectru transpus* cunoscut sub numele de *aliasing*. Această suprapunere de spectru distruge o parte din informația imaginii originale: frecvențele înalte (detalii ale imaginii) se pierd și apar interferențe (*aliases*) în regiunea frecvențelor joase.

Această interpretare este extensia la spațiul bidimensional a teoremei eșantionării. Conform teoremei eșantionării, o funcție continuă de o singură variabilă poate fi complet reprezentată printr-o mulțime de eșantioane calculate la intervale egale, dacă intervalul dintre două eșantioane este mai mic decât jumătate din perioada celei mai ridicate frecvențe componente a funcției.

Eșantionarea și redarea imaginii în grafica pe calculator se realizează prin calculul intensității în centrul pixelilor și asignarea acestei valori întregii zone spațiale care corespunde unui pixel.

Erorile de aliasing apar numai dacă spectrul frecvențelor spațiale ale imaginii conține valori mai mari decât jumătate din frecvența de eșantionare. O metodă de a reduce erorile de aliasing este aceea de a crește frecvența de eșantionare, adică de a crește rezoluția imaginii. Această metodă reduce, dar nu elimină zgomotul de aliasing, dat fiind că spectrul frecvențelor spațiale ale imaginii se poate extinde către infinit. Această situație este mai pregnantă în reprezentarea prin proiecție perspectivă a imaginilor tridimensionale. Din relația (4.10), prin care se calculează coordonatele punctelor prin proiecție perspectivă:  $x_N = d x_V / g z_V$ ,  $y_N = d y_V / h z_V$  rezultă că, pentru  $z_V \rightarrow \infty$ ,  $x_N \rightarrow 0$ ,  $y_N \rightarrow 0$ , adică toate punctele aflate de la distanțe foarte mari față de punctul de observare se proiectează foarte aproape de centrul ferestrei de proiecție. Prin proiecție perspectivă, obiectele aflate la distanțe foarte mari de punctul de observare produc elemente ale imaginii de dimensiuni foarte mici și deci cu o frecvență spațială care tinde către infinit.

Deci creșterea rezoluției imaginii nu elimină zgomotul de spectru transpus și, în plus, necesită creșterea corespunzătoare a rezoluției display-ului, care este limitată tehnologic și foarte costisitoare.

De aceea, pentru reducerea (eliminarea) erorilor de aliasing este necesară filtrarea imaginii înainte de eșantionare printr-un filtru trece-jos cu o frecvență de tăiere egală cu limita Nyquist (jumătate din frecvența de eșantionare). În felul acesta, frecvențele spațiale ale imaginii mai mari decât  $\omega_s/2$  sunt eliminate; detaliile care nu pot fi reprezentate la rezoluția disponibilă a sistemului grafic (număr de pixeli afișabili pe display) sunt pierdute, dar este eliminat și zgomotul de spectru transpus (*aliasing*) și imaginea poate fi eșantionată și reconstruită corect.

Pentru scene care modelează lumea reală, suprafețe de dimensiuni foarte mici se obțin, în general, prin proiecția perspectivă a obiectelor foarte depărtate de punctul de observare și de aceea eliminarea lor prin filtrarea anti-aliasing nu diminuează calitatea imaginii ci, din contra, o apropie mai mult de percepția vizuală reală, în care nu se pot vedea obiecte foarte depărtate.

Se poate remarca faptul că o imagine bună se poate obține numai prin combinarea celor două metode: creșterea, atât cât este posibil din punct de vedere



tehnologic, a rezoluției display-ului, prin care crește frecvența de eșantionare a imaginii, și filtrarea anti-aliasing, prin care se elimină detaliile cu frecvențe spațiale mai mari decât jumătate din frecvența de eșantionare, precum și zgomotul de spectru transpus (aliasing).

În grafica pe calculator au fost dezvoltate mai multe tehnici de filtrare a imaginilor pentru diminuarea efectelor de aliasing. Aceste tehnici, numite *tehnici de anti-aliasing*, încearcă să implementeze prin calcul cât mai eficient operația de filtrare a imaginilor. În esență, tehnicile de anti-aliasing reprezintă un compromis între simplitatea calculelor (și deci eficiența acestora) și rezultatul, din punct de vedere al percepției vizuale, al filtrării imaginilor.

Filtrarea, care este o multiplicare în domeniul frecvențelor, se poate calcula printr-o convoluție în domeniul spațial prin integrala:

$$f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i(x-u, y-v) h(u, v) du dv \quad (9.9)$$

unde:  $f(x, y)$  este imaginea filtrată;  
 $i(x, y)$  este imaginea nefiltrată  
 $h(u, v)$  este funcția pondere a filtrului.

Cele mai cunoscute metode de diminuare a aliasing-ului, care vor fi descrise în continuare, sunt:

- *Prefiltrarea imaginii.* În această metodă imaginea este filtrată cu un filtru trece-jos cu frecvența de tăiere egală cu intervalul Nyquist. Algoritmii dezvoltați în această tehnică se concentrează asupra aproximării cât mai bune și eficiente a funcției pondere a filtrului. Cele mai multe din sistemele grafice actuale implementează o variantă a prefiltrării, printr-o anumită aproximare a funcției pondere a filtrului.
- *Supraeșantionarea imaginii (postfiltrare).* Această metodă implementează convoluția în domeniul discret a unei imagini supraeșantionate.
- *Eșantionarea stocastică.* În această metodă se eșantionează imaginea într-o matrice (grilă) de eșantionare a cărei poziție este perturbată aleator față de poziția corespunzătoare centrului pixelilor. Eșantioanele obținute sunt folosite pentru determinarea intensității pixelilor printr-o filtrare de reconstrucție.

### 9.1.1 TEHNICA DE PREFILTRARE A IMAGINILOR

Un filtru trece-jos ideal unidimensional se definește prin funcția de transfer:

$$\begin{aligned} |H(\omega)| &= H_0, \quad -\arg H(\omega) = -\phi(\omega) = \omega t_0, \quad \tau_g = t_0 \text{ pentru } \omega \in (0, \omega_t), \\ |H(\omega)| &= 0, \quad -\arg H(\omega) = -\phi(\omega) = \pi, \quad \tau_g = 0 \text{ pentru } \omega \in (\omega_t, \infty) \end{aligned} \quad (9.10)$$



Funcția pondere a unui filtru,  $h(t)$ , este răspunsul filtrului la funcția impuls Dirac. Pentru calculul răspunsului filtrului trece-jos la funcția impuls Dirac, se folosește relația transformatei Fourier inverse:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) e^{j\omega t} d\omega \quad (9.11)$$

Înlocuind expresia lui  $H(\omega)$  în relația (9.11) se obține:

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} H_0 e^{-j\omega t_0} e^{j\omega t} d\omega = \frac{H_0}{\pi} \int_0^{\omega_t} \cos \omega(t - t_0) d\omega \quad (9.12)$$

Efectuând integrala rezultă:

$$h(t) = \frac{H_0}{\pi} \frac{\sin \omega_t(t - t_0)}{t - t_0} \quad (9.13)$$

Dacă se notează  $v_t = \omega_t(t - t_0)$ , atunci:

$$h(v_t) = \frac{H_0 \omega_t}{\pi} \frac{\sin v_t}{v_t} = \frac{H_0 \omega_t}{\pi} \text{sinc } v_t \quad (9.14)$$

Funcția pondere a unui filtru trece-jos unidimensional ideal este reprezentată în fig. 9.3.

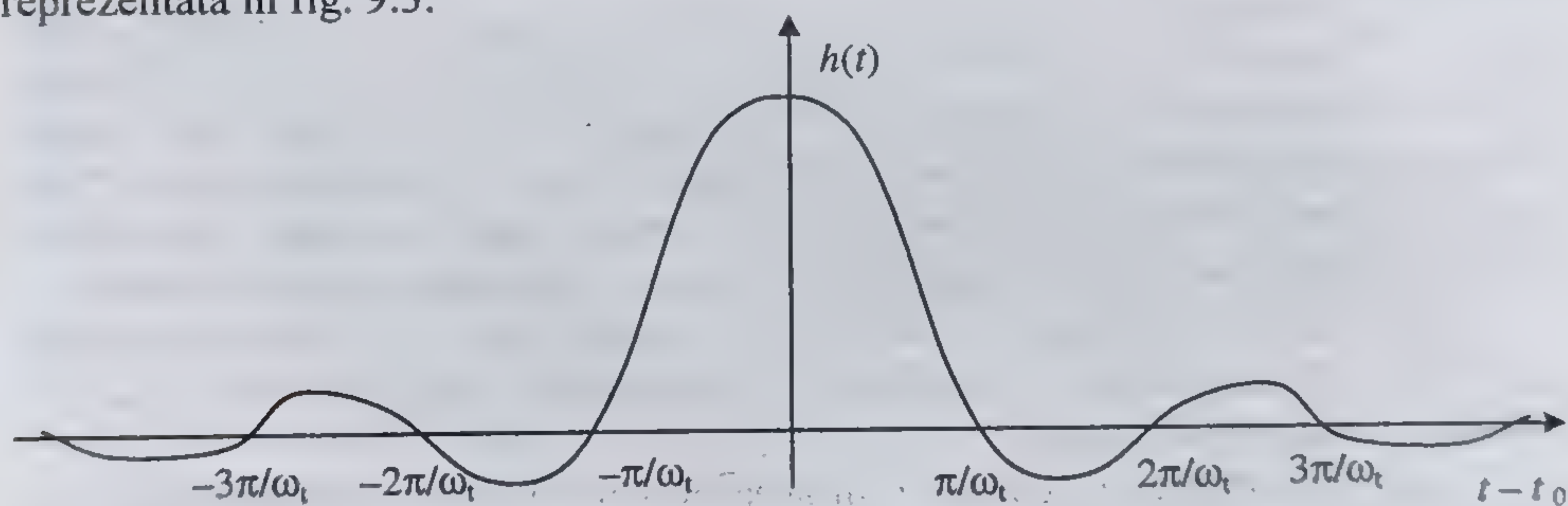


Fig. 9.3 Funcția pondere a unui filtru trece-jos unidimensional ideal.

Pentru filtrarea imaginilor se folosesc filtre bidimensionale, ale căror caracteristici se pot deduce prin extinderea caracteristicilor filtrelor unidimensionale. Problema filtrării imaginilor este de a determina funcția pondere  $h(n,m)$  a unui filtru cu caracteristica  $H(u,v)$  de filtru trece-jos bidimensional. Această funcție nu se poate determina exact și de aceea se folosesc funcții care aproximează cât mai apropiat și eficient, funcția pondere a unui filtru bidimensional trece-jos.

Tehnicile de prefiltrare pentru diminuarea zgomotului de aliasing se bazează pe aproximarea funcției pondere a filtrului trece jos. În 1981 Crow a testat mai multe funcții de filtrare a imaginilor și a arătat că imaginea reconstruită este cu



atât mai bună cu cât funcția folosită este mai apropiată de funcția pondere a filtrului trece-jos ideal [Crow81]. În fig. 9.4 sunt reprezentate diferite funcții de convoluție în spațiul unidimensional care se pot folosi în prefiltrare. Funcțiile în spațiul bidimensional folosite sunt extensia bidimensională a acestora.

Cea mai puțin costisitoare funcție de convoluție care se folosește în prefiltrarea anti-aliasing este funcția dreptunghiulară pe o distanță interpixel (fig. 9.4 (a)). Această funcție reprezintă o aproximare destul de grosieră a funcției pondere a unui filtru trece-jos ideal. Extensia acestei funcții în spațiul bidimensional este numită fereastră Fourier pe o distanță interpixel, iar integrala de convoluție devine o medie ponderată a intensităților tuturor suprafețelor care acoperă pixelul respectiv.

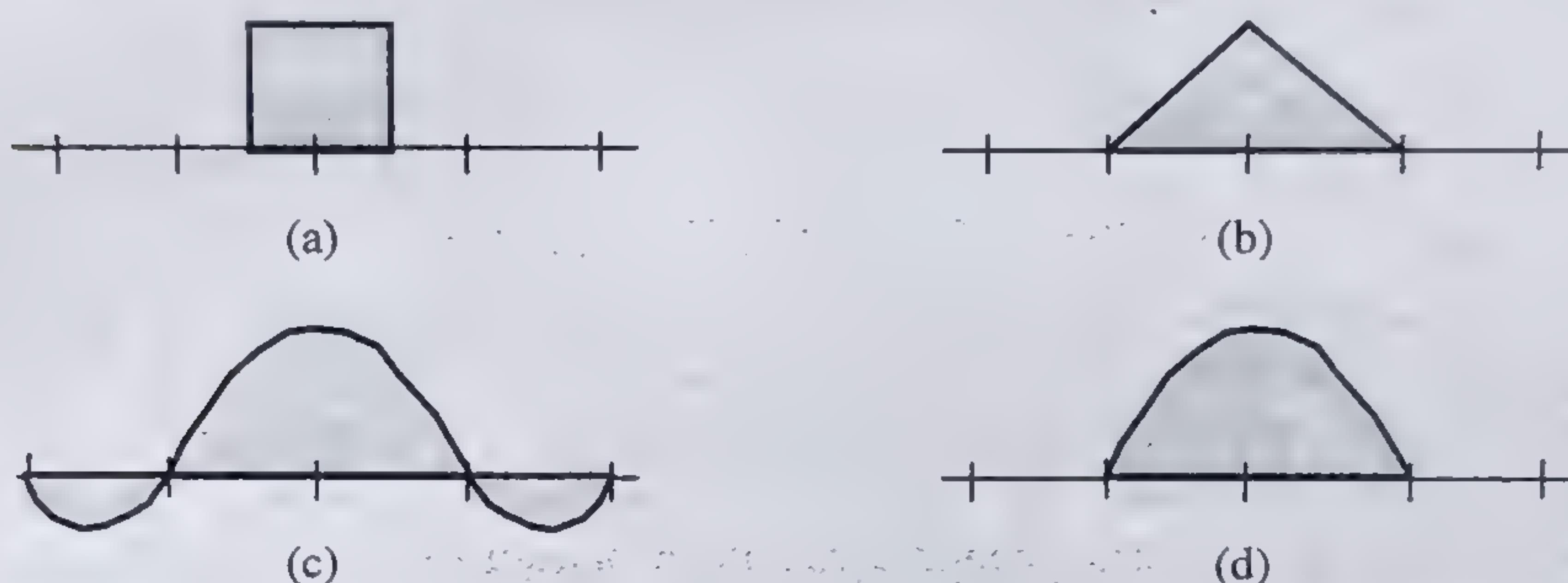


Fig. 9.4 Funcții de convoluție:

- (a) dreptunghi pe o distanță interpixel; (b) triunghi peste două distanțe interpixel;
- (c) funcția  $\sin x / x$  pe patru distanțe interpixel;
- (d) lobul central al funcției  $\sin x / x$  pe două distanțe interpixel.

**Algoritmul de prefiltrare al lui Catmull.** Algoritmul de prefiltrare folosind fereastră Fourier, dezvoltat inițial de Catmull [Cat78], calculează intensitatea fiecărui pixel folosind aria fiecărui *fragment* de suprafață vizibilă în acel pixel ca pondere în însumarea intensităților acestora. Acest mod de calcul este echivalent cu convoluția imaginii cu o funcție pondere a filtrului spațial bidimensional de formă paralelipipedică; valoare intensității calculată prin ponderare este atribuită apoi pixelului. Pentru calculul intensității prin ponderare se efectuează operații la nivel de subpixel în spațiul continuu al imaginii (fig. 9.5).

Pentru calculul intensității unui pixel se execută o operație de decupare a fiecărei suprafețe folosind ca fereastră de decupare pătratul de delimitare a pixelului. Dacă fragmentele mai multor poligoane se suprapun, atunci se sortează în funcție de adâncime și se decupează din nou, unul față de altul, pentru a se decide fragmentele de poligoane vizibile în interiorul pixelului. Fie ariile  $A_1, A_2, \dots, A_n$  ale fragmentelor vizibile în pixel,  $I_1, I_2, \dots, I_n$  intensitățile corespunzătoare acestora și  $I_B$  intensitatea de bază a pixelului (setată la ștergerea imaginii). Intensitatea rezultantă a pixelului este:



$$I = I_1 A_1 + I_2 A_2 + \dots + I_n A_n + I_B (1 - A_1 - A_2 - \dots - A_n) \quad (9.15)$$

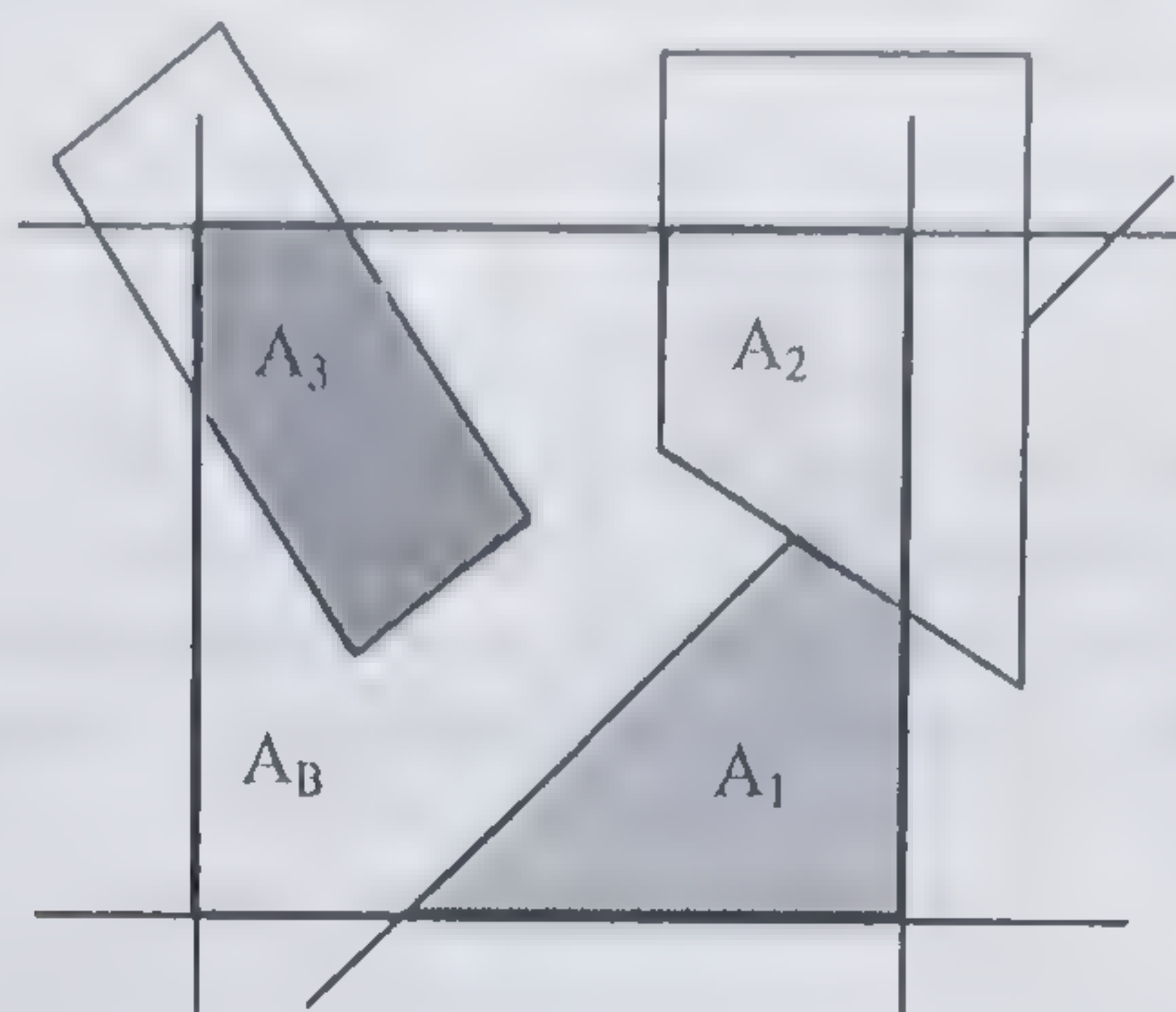


Fig. 9.5 Calculul intensității unui pixel prin ponderare:

$$I = I_1 A_1 + I_2 A_2 + I_3 A_3 + I_B A_B$$

Algoritmul lui Catmull poate fi încorporat în algoritmul de generare a poligoanelor prin baleiere pe linii. Pentru fiecare linie de baleiere se disting trei tipuri de pixeli: pixeli marginali, stânga și dreapta și pixeli interiori (fig. 9.6). Pentru pixelii marginali se ponderează intensitățile cu aria fragmentului corespunzător obținut prin decupare relativ la pătratul de delimitare a pixelului. Pentru pixelii interiori poligonului (acoperiți complet de un poligon) nu se mai calculează fragmente prin decupare și se atribuie intensitatea calculată prin interpolare, conform modelului de umbrire folosit.

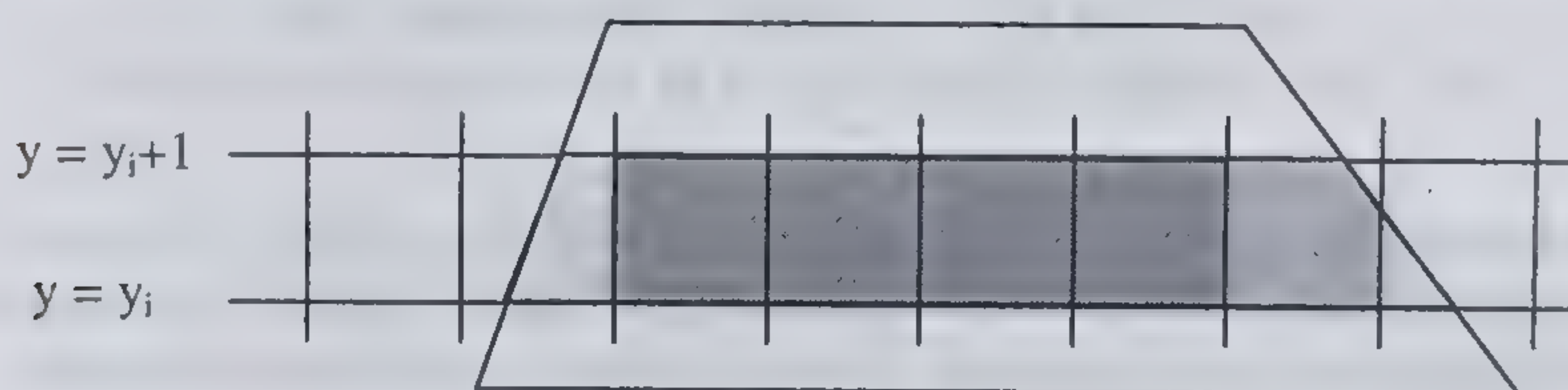


Fig. 9.6 Operația de baleiere pe linii și prefiltrare anti-aliasing.

Algoritmul original dezvoltat de Catmull este deosebit de costisitor ca timp de execuție. Calculul decupării poligoanelor la limitele pixelului ca și media ponderată a fragmentelor necesită un mare număr de operații în virgulă flotantă.

Pe baza acestui algoritm s-au dezvoltat ulterior alți algoritmi mai eficienți, care simplifică calculele de ponderare prin aproximarea fragmentele suprafețelor ce acoperă un pixel cu un număr întreg de subpixeli.

**Algoritmul de prefiltrare al lui Carpenter.** Algoritmul dezvoltat de Carpenter [Car84] folosește o tehnică combinată de filtrare anti-aliasing și Z-buffer, tehnică cunoscută sub numele de A-buffer (*anti-aliased, area-averaged, accumulation buffer*).



Principalul avantaj al acestui algoritm este acela că evită calculele în virgulă flotantă pentru ponderarea ariilor fragmentelor vizibile. Fiecare pixel este divizat într-un număr de  $k \times k$  subpixeli (fig. 9.7). Aria unui fragment al unui poligon se aproximează cu numărul întreg de subpixeli acoperiți de poligonul respectiv. Un subpixel este considerat acoperit de un poligon dacă este acoperit în proporție mai mare de 0,5 din aria sa. Cu cât numărul de subpixeli în care este divizat pixelul este mai mare, cu atât precizia de calcul a ponderilor intensităților este mai mare și calitatea imaginii obținute este mai bună.

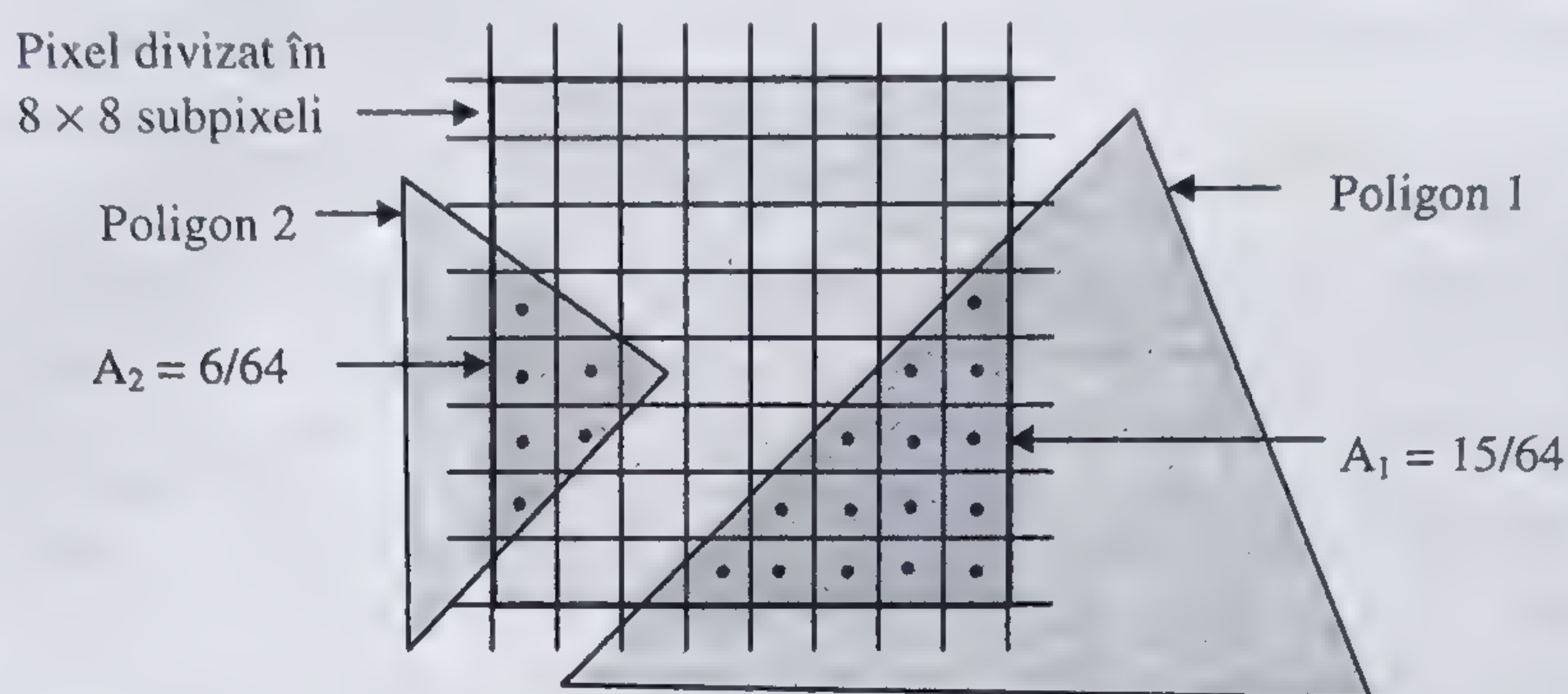


Fig. 9.7 Ponderarea ariilor fragmentelor poligoanelor prin divizarea pixelului.

În majoritatea documentațiilor tehnice calitatea anti-aliasing-ului oferit de un sistem grafic (accelerator grafic) este dată prin numărul  $k \times k$  de subpixeli. Cu cât numărul de subpixeli este mai mare, cu atât ponderile (ariile fragmentelor) se calculează mai exact și filtrarea anti-aliasing este mai eficientă.

Detaliile de definire a bufferului de acumulare și de calcul al ponderilor atribuite fragmentelor suprafețelor sunt complexe, dependente de implementarea specifică și dificil de descris la modul general. În subcapitolul următor sunt descrise modalitățile de filtrare anti-aliasing oferite de biblioteca OpenGL.

### 9.1.2 TEHNICA DE SUPRAEȘANTIONARE A IMAGINII

Tehnica de supraeșantionare (sau postfiltrare) pentru reducerea zgomotului de aliasing constă din trei etape de calcul:

- Imaginea din spațiul bidimensional continuu este eșantionată printr-o matrice de eșantionare de dimensiune  $(k \times n) \times (k \times n)$ , unde  $n \times n$  reprezintă numărul de pixeli care se pot afișa în fereastra de afișare, iar  $k \times k$  reprezintă factorul de supraeșantionare. În practică acest lucru înseamnă generarea imaginii cu o rezoluție de  $k \times k$  ori mai mare decât rezoluția displayului.
- Imaginea eșantionată este filtrată folosind un filtru trece-jos cu frecvența de tăiere egală cu limita Nyquist a displayului.
- Imaginea filtrată este reeșantionată la rezoluția displayului.



Această tehnică înseamnă generarea unei imagini virtuale cu o rezoluție mai mare decât rezoluția displayului, urmată de reducerea acestei rezoluții la valoarea rezoluției displayului prin filtrare. Intensitatea fiecărui pixel se determină din valorile unui număr de  $k \times k$  subpixeli din imaginea virtuală. Acest procedeu este prezentat în fig. 9.8. În acest exemplu, imaginea este generată cu o rezoluție de  $3 \times 3$  ori mai mare decât rezoluția imaginii afișate folosind procedurile cunoscute de conversie prin baleiere pe linii a poligoanelor, combinate cu umbrire și Z-buffer. Grupuri de  $3 \times 3$  subpixeli din imaginea virtuală sunt apoi folosiți pentru generarea intensității fiecărui pixel din imaginea reală prin ponderarea intensității fiecărui subpixel cu valorile corespunzătoare ale filtrului, însumare și normalizare.

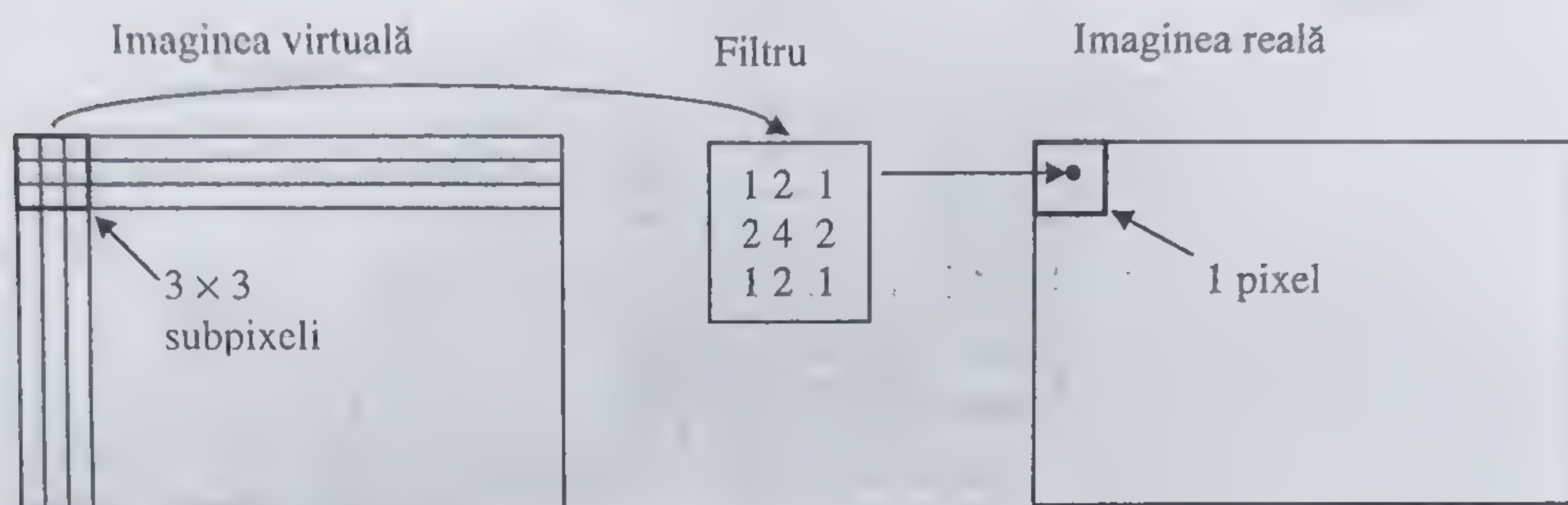


Fig. 9.8 Supraeșantionarea și filtrarea imaginii.

Această metodă de ponderare este mai performantă decât medierea corespunzătoare ferestrei Fourier și este cunoscută sub numele de ponderare în fereastra Barlett. Fereastra Barlett poate avea diferite dimensiuni (de exemplu,  $3 \times 3$ ,  $5 \times 5$  sau  $7 \times 7$ ) în funcție de factorul de supraeșantionare. Ponderile corespunzătoare ale ferestrei Barlett asigură contribuția subpixelilor la intensitatea finală a pixelului cu atât mai substanțială cu cât sunt mai apropiați de centrul ferestrei (tabelul 9.1).

Tabelul 9.1

Ferestre Barlett folosite în postfiltrarea imaginilor supraeșantionate

$3 \times 3$			$5 \times 5$					$7 \times 7$						
1	2	1	1	2	3	2	1	1	2	3	4	3	2	1
2	4	2	2	4	6	4	2	2	4	6	8	6	4	2
1	2	1	3	6	9	6	3	3	6	9	12	9	6	3
			2	4	6	4	2	4	8	12	16	12	8	4
			1	2	3	2	1	3	6	9	12	9	6	3
								2	4	6	8	6	4	2
								1	2	3	4	3	2	1



Operația de filtrare pentru reducerea zgomotului de aliasing, definită prin integrala de convoluție din relația (9.9), se execută prin aproximare în tehnica de supraeșantionare. Atunci când se efectuează ponderarea, imaginea este deja eșantionată și nu este disponibilă funcția continuă  $i(x, y)$ , ci numai eșantioane ale acesteia. Îmbunătățirea aproximării se obține prin creșterea factorului de supraeșantionare, dar acest lucru este costisitor, atât ca volum de memorie necesar (Z-bufferul crește și el cu același factor de supraeșantionare) cât și ca timp de execuție. Această aproximare este asemănătoare cu aproximarea introdusă în tehnica de prefiltrare prin divizarea pixelului în subpixeli, astfel că distincția între prefiltrare și postfiltrare este oarecum forțată.

### 9.1.3 EȘANTIONAREA STOCASTICĂ

Principiul de bază al acestei metode este de a perturba în mod aleator poziția punctelor de eșantionare. În acest fel, frecvențele înalte ale imaginii, peste limita Nyquist, sunt transformate în zgomot, care apare în spectrul eșantioanelor în locul zgomotului de spectru transpus, și poate fi eliminat prin filtrare.

În fig. 9.9 (a) o undă sinusoidală este eșantionată cu o frecvență mai mare decât limita Nyquist. Perturbația punctului de eșantionare introduce o eroare în amplitudinea eșantioanelor, care apare ca un zgomot în spectrul imaginii eșantionate.

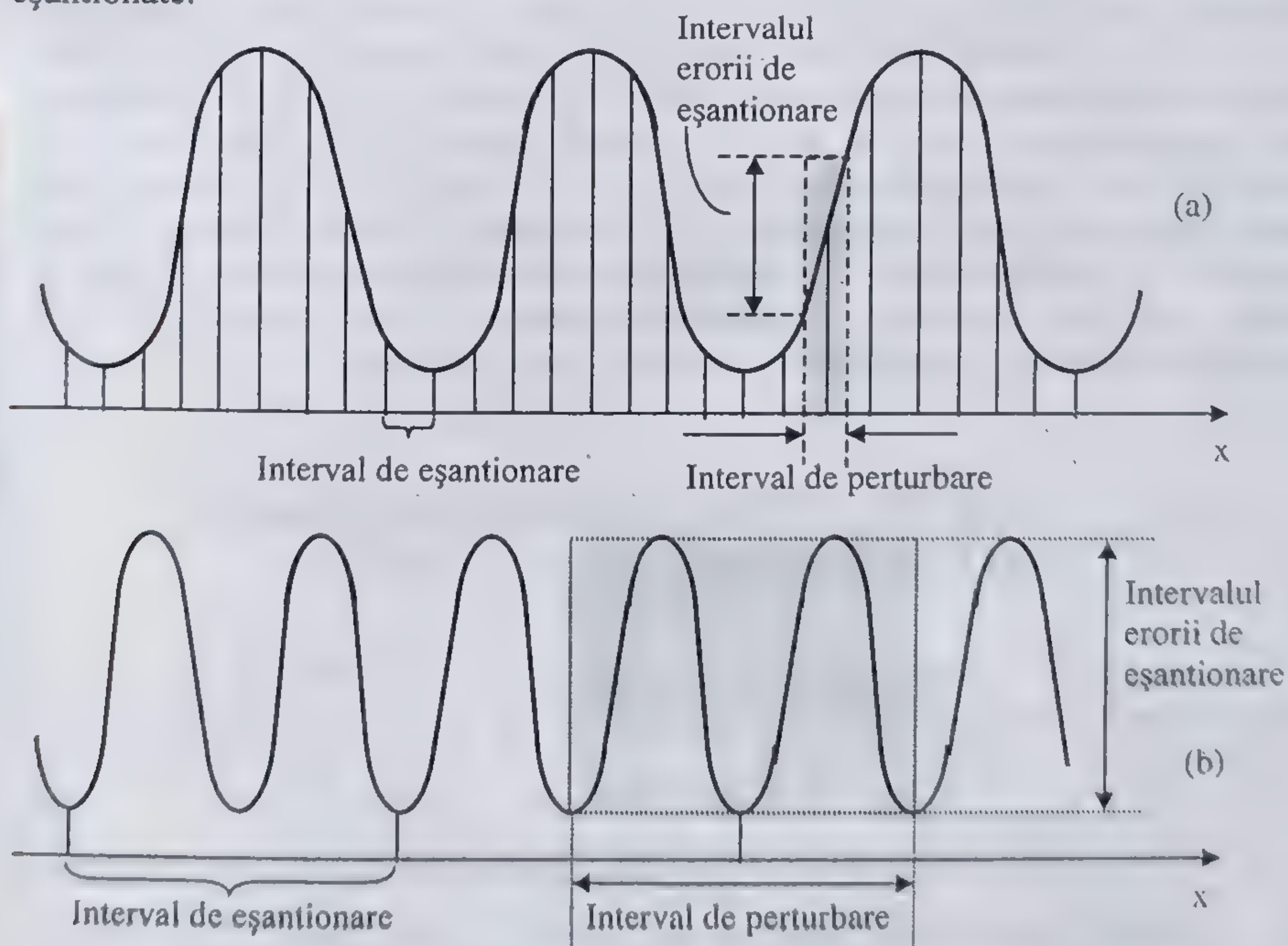


Fig. 9.9 Eșantionarea stocastică a unei unde sinusoidale:  
 (a) frecvența de eșantionare mai mare decât limita Nyquist;  
 (b) frecvența de eșantionare mai mică decât limita Nyquist.



În fig. 9.9 (b) unda sinusoidală este eșantionată cu o frecvență mai mică decât limita Nyquist. Perturbația punctului de eșantionare se extinde într-un interval egal cu intervalul de eșantionare, iar eroarea de eșantionare rezultată este o eroare aleatoare. În acest fel, eroarea de spectru transpus (aliasing) care ar fi rezultat prin eșantionarea uniformă a acestei unde este înlocuită cu un zgomot aleator, care poate fi eliminat prin filtrare.

În sebcapitolul următor este prezentată modalitatea de implementare a tehnicii de anti-aliasing prin eșantionare stocastică folosind un buffer de acumulare (A-buffer) pentru eliminarea zgomotului aleator rezultat.

Implementarea tehnicilor de anti-aliasing diferă foarte mult de la un sistem grafic la altul. O imagine de calitate (fără zgomot de spectru transpus – aliasing) necesită atât rezoluții mari ale imaginii, care permit creșterea frecvenței de eșantionare, cât și filtrarea imaginii, prin care se limitează spectrul imaginii la o valoare care asigură eșantionarea și reconstrucția fără zgomot (sau cu zgomot redus) a imaginii.

Așa cum se poate observa din descrierile precedente, toți algoritmii folosiți pentru diminuarea efectului de aliasing necesită resurse de memorie și de calcul considerabile și, cu cât se admit aproximații mai grosiere, cu atât scade calitatea imaginii datorită zgomotului de spectru transpus.

În sistemele grafice folosite în diferite aplicații de realitate virtuală, asigurarea unei filtrări anti-aliasing de calitate este o condiție foarte importantă, mai ales în simulatoarele de antrenament. În cursul antrenamentelor de zbor, mișcarea dezordonată (*crawl*) a obiectelor, ca și apariția sau dispariția obiectelor mici, datorată aliasing-ului, au un efect negativ puternic asupra deprinderilor pe care trebuie să le însușească piloții. De aceea, s-au făcut eforturi deosebite pentru implementarea hardware a algoritmilor de anti-aliasing, alături de implementarea hardware a Z-bufferului, sau a texturării. Calitatea implementării procedurilor de anti-aliasing este un criteriu important de selecție a unui sistem grafic, și, bineînțeles, reprezintă un procent important din costul acestuia.

## 9.2 FUNCȚII OpenGL PENTRU REDARE ANTI-ALIASING

Biblioteca OpenGL pune la dispoziție două modalități de redare anti-aliasing: prin combinarea culorilor (*blending*) și prin buffer de acumulare (A-buffer).

### 9.2.1 ANTI-ALIASING PRIN COMBINAREA CULORILOR

Această metodă dă rezultate de netezire numai dacă se renunță la mecanismul de Z-buffer (bufferul de adâncime) și se desenează suprafețele în ordinea în care sunt transmise bibliotecii. În această situație, culoarea unui pixel acoperit parțial de un poligon se obține prin combinarea culorii poligonului cu



culoarea pixelului, aflată în bufferul de culoare. În modul RGBA, OpenGL multiplică componenta  $A$  (*alpha*) a culorii poligonului cu ponderea de acoperire (raportul dintre aria acoperită de fragmentul poligonului și aria pixelului). Această valoare poate fi folosită pentru ponderarea culorii în fiecare pixel prin combinare cu factorul `GL_SRC_ALPHA` pentru sursă și factorul `GL_ONE_MINUS_SRC_ALPHA` și pentru destinație. Pentru efectuarea acestor calcule, mai este necesară validarea anti-aliasing-ului prin apelul funcției `glEnable()` cu unul din argumentele `GL_POINT_SMOOTH`, `GL_LINE_SMOOTH`, `GL_POLYGON_SMOOTH` pentru puncte, linii și, respectiv, poligoane.

### ■ Exemplul 9.1

Programul din acest exemplu este folosit pentru generarea imaginilor din fig. 9.10 în care se desenează două triunghiuri cu și fără anti-aliasing. Funcțiile `Init()` și `Display()` ale programului dezvoltat sub GLUT sunt următoarele:

```
void Init(){
    glClearColor(1.0,1.0,1.0,1.0);
    glLineWidth (1.5);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}

void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    glPushMatrix();
    glTranslated(0.0,0.0,-25.0);
    // Triunghiuri cu aliasing
    glDisable(GL_BLEND);
    glDisable(GL_LINE_SMOOTH);
    glDisable(GL_POLYGON_SMOOTH);
    glPushMatrix();
    glTranslated(-8,0,0.0);
    glBegin(GL_LINE_LOOP);
        glVertex3d(-3,-3,0);
        glVertex3d( 3,-2,0);
        glVertex3d( 2,3,0);
    glEnd();
    glPopMatrix();
    glPushMatrix();
    glTranslated(-2,0,0.0);
    glBegin(GL_POLYGON);
        glVertex3d(-3,-3,0);
        glVertex3d( 3,-2,0);
        glVertex3d( 2,3,0);
    glEnd();
    glPopMatrix();
    // Triunghiuri anti-aliasing
    glEnable(GL_BLEND);
    glEnable(GL_LINE_SMOOTH);
```

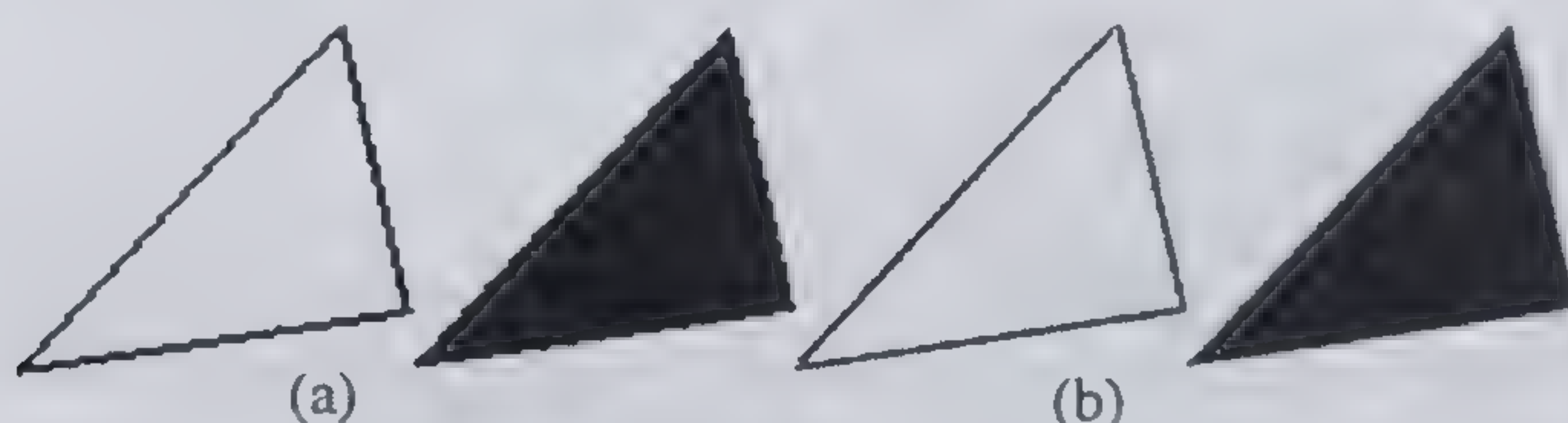


```

glEnable(GL_POLYGON_SMOOTH);
glPushMatrix();
glTranslated(4, 0, 0.0);
glBegin(GL_LINE_LOOP);
    glVertex3d(-3, -3, 0);
    glVertex3d( 3, -2, 0);
    glVertex3d( 2, 3, 0);
glEnd();
glPopMatrix();
glPushMatrix();
glTranslated(10, 0, 0.0);
glBegin(GL_POLYGON);
    glVertex3d(-3, -3, 0);
    glVertex3d( 3, -2, 0);
    glVertex3d( 2, 3, 0);
glEnd();
glPopMatrix();
glPopMatrix();
glutSwapBuffers();
)

```

Deoarece anti-aliasing-ul se realizează prin combinarea culorilor, este necesară și validarea combinării culorilor (`glEnable(GL_BLEND)`) pentru poligoanele cu anti-aliasing.



**Fig. 9.10** (a) Două poligoane cu aliasing.  
(b) Aceleași poligoane reprezentate cu anti-aliasing.

Acest mod de filtrare anti-aliasing este foarte simplist și nu poate fi folosit decât pentru suprafețe sau linii care nu se suprapun și pot fi redată fără eliminarea suprafețelor ascunse, așa cum a fost situația în acest exemplu.

Dacă se redau obiecte tridimensionale complexe sau mai multe obiecte în scenă, este necesar să fie asigurată eliminarea suprafețelor ascunse. Combinarea culorilor nu funcționează corect împreună cu mecanismul de Z-buffer, iar generarea imaginilor fără eliminarea suprafețelor ascunse este lipsită de realism.

Dacă se invalidează bufferul de adâncime, eliminarea suprafețelor ascunse trebuie asigurată printr-un alt mecanism, de exemplu prin ordonarea suprafețelor după adâncime. Dar eliminarea suprafețelor ascunse prin ordonarea suprafețelor după adâncime este un procedeu costisitor și care nu poate fi implementat pentru scene complexe, cu un număr mare de obiecte. Soluția acceptabilă ca performanțe și rezultate pentru redarea anti-aliasing a scenelor complexe este folosirea unui buffer de acumulare (A-buffer).



### 9.2.2 ANTI-ALIASING PRIN ACUMULARE

Biblioteca OpenGL implementează un buffer de acumulare care este folosit atât pentru redarea anti-aliasing a scenelor, cât și pentru crearea altor efecte vizuale, ca iluminarea cu surse multiple de lumină sau estomparea imaginii unui obiect datorită mișcării acestuia (*motion blur*).

Bufferul de acumulare se definește cu aceeași dimensiune ca și bufferul de culoare și bufferul de adâncime. În fiecare locație (x, y) a bufferului de acumulare se memorează culoarea unui pixel, corespunzător pixelului cu aceeași adresă (x, y) în bufferul de culoare. Asupra bufferului de acumulare se pot efectua operații prin apelul funcției:

```
void glAccum(GLenum op, GLfloat value);
```

Parametrul op selectează operația, iar parametrul value este un număr care este folosit în unele operații. Operațiile posibile sunt GL\_ACCUM, GL\_LOAD, GL\_RETURN, GL\_ADD și GL\_MULT.

- GL\_ACCUM citește fiecare pixel din bufferul curent de citire, selectat anterior cu funcția glReadBuffer(), multiplică valorile R,G,B,A ale pixelului cu valoarea value și rezultatul îl adună la valoarea corespunzătoare (cu aceeași adresă de pixel) din bufferul de acumulare.
- GL\_LOAD se execută asemănător cu operația GL\_ACCUM, cu deosebirea că valorile rezultate înlocuiesc valorile existente în bufferul de acumulare.
- GL\_RETURN preia valorile culorilor fiecărui pixel din bufferul de acumulare, le multiplică cu valoarea value, iar rezultatul îl înscrie în buferul de scriere selectat anterior printr-o funcție glDrawBuffer().
- GL\_ADD și GL\_MULT adună sau înmulțește valoarea componentelor R,G,B,A a fiecărui pixel din bufferul de acumulare cu valoarea value, rezultatul fiind depus înapoi în bufferul de acumulare. Pentru operația GL\_MULT, value se limitează în intervalul [- 1.0, 1.0].

Înainte de începerea fiecărei operații de acumulare bufferul de acumulare se șterge prin apelul funcției glClear(GL\_ACCUM\_BUFFER\_BIT). Culoarea de ștergere a bufferului de acumulare trebuie să fie (0,0,0,0), și se setează la inițializare prin apelul funcției glClearAccum(0.0, 0.0, 0.0, 0.0).

Bufferul de acumulare se folosește pentru implementarea anti-aliasing-ului prin eșantionare stocastică. Imaginea este eșantionată în poziții de eșantionare care sunt perturbate aleator față de centrul pixelului, pe o distanță egală cu dimensiunea pixelului. Imaginile succesive obținute pentru toate pozițiile de eșantionare sunt mediate pentru fiecare componentă de culoare a fiecărui pixel, rezultând imaginea finală, din care a fost eliminat zgomotul aleator datorat perturbării pozițiilor de eșantionare. Fiecare imagine se creează într-un buffer de culoare și apoi se adaugă ponderat la bufferul de acumulare (inițial șters) prin funcția



`glAccum(GL_ACCUM, 1/n)`, unde  $n$  este numărul de poziții succesive de eșantionare. Imaginea rezultată în bufferul de eșantionare este transferată în bufferul de culoare prin funcția `glAccum(GL_RETURN, 1.0)`. La generarea fiecărei imagini corespunzătoare unei poziții de eșantionare se execută testul de adâncime (Z-buffer), deci imaginea finală elimină suprafețele ascunse.

Așa cum se vede în Planșa 6, efectele de aliasing sunt mult diminuate prin acest procedeu, dar, dacă este implementat soft, timpul de calcul este foarte mare, cel puțin de  $n$  ori mai mare decât timpul necesar pentru aceeași imagine redată fără anti-aliasing. Implementarea hardware a tehnicii de anti-aliasing este absolut necesară în grafica interactivă. Programul prin care s-a realizat această imagine este dat în exemplul următor.

### ■ Exemplul 9.2

Redarea anti-aliasing a obiectelor cu umbrire și eliminarea suprafețelor ascunse se poate implementa folosind bufferul de acumulare astfel:

```
#include <GL/glut.h>
#define ACSIZE 16 // numărul de esantionari
static float W = 6; // dimensiune maxima fereastră
static float WX, WY; // dimensiuni fereastră
static float N = -10; // distanta de vizualizare near
static float F = 10; // distanta de vizualizare far
void Init(void) {
    GLfloat ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat position[] = { 1.0, 0.2, 1.0, 0.0 };
    GLfloat mat_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat mat_diffuse[] = { 0.9, 0.2, 0.1, 1.0 };
    GLfloat mat_specular[] = { 0.9, 0.6, 0.6, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glClearAccum(0.0, 0.0, 0.0, 0.0);
    glClearColor(0.8, 0.8, 0.8, 1.0);
}
GLfloat jitter16[][2] = {
    {0.375, 0.4375}, {0.625, 0.0625},
    {0.875, 0.1875}, {0.125, 0.0625},
```



```

    {0.375, 0.6875}, {0.875, 0.4375},
    {0.625, 0.5625}, {0.375, 0.9375},
    {0.625, 0.3125}, {0.125, 0.5625},
    {0.125, 0.8125}, {0.375, 0.1875},
    {0.875, 0.6875}, {0.875, 0.0625},
    {0.125, 0.3125}, {1.625, 0.8125}
};

void Display() {
    int i;
    glClear(GL_ACCUM_BUFFER_BIT);
    GLint viewport[4];
    glGetIntegerv(GL_VIEWPORT, viewport);
    // Redarea cu anti-aliasing a obiectului
    for (i = 0; i < (ACSIZE); i++) {
        glPushMatrix();
        glTranslatef(jitter16[i][0]*2*WX/viewport[2],
                     jitter16[i][1]*2*WY/viewport[3], 0.0);
        glTranslated(1.5, -1, -8);
        glRotatef(15.0, 1.0, 0.0, 0.0);
        glRotatef(20.0, 0.0, 1.0, 0.0);
        glClear(GL_DEPTH_BUFFER_BIT |
                GL_COLOR_BUFFER_BIT);
        glutSolidTeapot(1.0);
        glPopMatrix();
        glAccum(GL_ACCUM, 1.0/(ACSIZE));
    }
    glAccum(GL_RETURN, 1.0);
    // Obiectul redat fara anti-aliasing
    glPushMatrix();
    glTranslated(-1.5, -1, -8);
    glRotatef(15.0, 1.0, 0.0, 0.0);
    glRotatef(20.0, 0.0, 1.0, 0.0);
    glutSolidTeapot(1.0);
    glPopMatrix();
    glFlush();
}

void Reshape(int w, int h) {
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    if (w <= h) {
        WX = W/2; WY = W*(GLfloat) h/(GLfloat) (w*2);
    }
    else {
        WX = W*(GLfloat) w/(GLfloat) (h*2); WY = W/2;
    }
    glOrtho(-WX, WX, -WY, WY, N, F);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

```



```

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|
                        GLUT_DEPTH|GLUT_ALPHA|GLUT_ACCUM);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Acumulare");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```

În funcția de inițializare `Init()` se definește sistemul de iluminare (surse de lumină și materiale), se stabilesc culorile de ștergere ale bufferului de culoare și de acumulare și se validează bufferul de adâncime. Unele din comenzile din inițializare nu au mai fost introduse, dacă s-au folosit valorile implicite ale bibliotecii OpenGL. De exemplu, în execuția cu un singur buffer de culoare (stabilită la inițializare), bufferul de citire și scriere curent este implicit bufferul de culoare `GL_FRONT`, și nu s-au mai apelat funcțiile `glReadBuffer()` și `glDrawBuffer()`.

Perturbarea poziției de eșantionare se obține prin perturbarea poziției centrului porții de afișare. În acest exemplu s-a folosit o proiecție ortografică în care transformările inverse de la poarta de afișare în sistemul universal se execută mai simplu. Fereastra este definită ca o fereastră simetrică, de dimensiuni  $2 \cdot WX$ ,  $2 \cdot WY$ , calculate în funcția `Reshape()`. Perturbația grilei de eșantionare se obține prin modificarea poziției centrului porții cu valorile aleatoare ( $\epsilon_x$ ,  $\epsilon_y$ ) care se citesc din tabelul `jitter`. Acest tabel conține perechi de valori în număr egal cu numărul de acumulări dorite (`ACSIZE`), care sunt poziții aleatoare pe suprafața unui pixel.

Perturbațiile în poarta de afișare se transformă în perturbare a poziției centrului ferestrei de vizualizare prin scalare cu factorii de scalare  $2 \cdot WX / \text{viewport}[2]$ ,  $2 \cdot WY / \text{viewport}[3]$ . Aceste deplasări ale poziției centrului ferestrei de vizualizare sunt echivalente cu deplasări în sens invers ale obiectelor scenei. În proiecția ortografică, deplasarea obiectelor care conduce la deplasarea centrului porții de afișare și, deci, la perturbarea poziției grilei de eșantionare se obține prin instrucțiunea:

```

glTranslatef(jitter16[i][0]*2*WX/viewport[2],
             jitter16[i][1]*2*WY/viewport[3],0.0);

```

În proiecție perspectivă, deplasarea obiectelor trebuie să fie calculată în funcție de matricea de proiecție perspectivă. Aceste calcule sunt propuse ca exercițiu pentru cititori.



## TEXTURAREA

Texturarea este o tehnică importantă de creștere a realismului scenelor virtuale, prin care se generează imagini ale obiectelor tridimensionale mult mai interesante și mai complexe. Prin texturare, se modulează culoarea suprafețelor obiectelor folosind o imagine repetitivă (textura), obținându-se un efect asemănător aceleia de aplicație (lipire) a unui desen pe o suprafață.

În decursul dezvoltării tehnicilor de texturare, au fost propuse diferite modalități de creare a impresiei de realism a suprafețelor, de exemplu prin perturbația normalelor suprafețelor, sau prin variații ale iluminării mediului ambiant al obiectelor. În momentul actual, metoda cea mai larg răspândită, cu numeroase implementări hardware în sistemele grafice, este metode de modulare a culorii suprafețelor obiectelor folosind imagini de texturi. Texturile care modulează culoarea suprafețelor sporesc informația vizuală prezentată prin sugerarea materialului suprafeței. De exemplu, prin aplicația repetată a imaginii unei cărămizi (obținută prin scanarea unei fotografii a unui perete real), un singur dreptunghi capătă aspectul unui perete întreg (Planșa 7(a)). Desenarea unui astfel de perete folosind câte un poligon pentru fiecare cărămidă este extrem de inefficientă și imaginea obținută este nerealistă, deoarece fiecare cărămidă ar fi prea netedă și regulată. De asemenea, prin texturare se poate aplica pe o suprafață o imagine care este practic imposibil să fie generată prin suprafețe poligonale (Planșa 7(b)).

### 10.1 APLICAȚIA TEXTURILOR

Pentru texturarea obiectelor prin modularea culorii suprafețelor trebuie să se definească următorii parametri:

- Imaginea texturii aplicate.
- Modul de aplicație a texturii pe suprafața unui obiect.
- Modul de diminuare (eliminare) a efectelor de aliasing care apar în aplicarea texturilor (filtrarea texturilor).



Texturile se definesc ca imagini bidimensionale care se aplică suprafețelor obiectelor, sau ca un câmp tridimensional, și atunci culoarea obiectului se determină prin intersecția suprafețelor sale cu acest câmp tridimensional al texturii.

### 10.1.1 APLICAȚIA TEXTURILOR BIDIMENSIONALE

Textura bidimensională este o imagine definită într-un spațiu bidimensional (numit *spațiul texturii*)  $T(s, t)$ , în intervalul  $[0,1]$  pe ambele axe de coordonate  $s$  și  $t$ . Textura este alcătuită din elementele componente discrete care sunt dreptunghiuri (sau pătrate) numite *texeli*. Fiecare imagine de textură este definită prin rezoluția ei, dată ca număr de texeli pe cele două coordonate. De exemplu, se folosesc imagini de textură de  $256 \times 256$  texeli,  $512 \times 512$ , etc.

Aplicația texturii bidimensionale unui obiect constă din parametrizarea suprafețelor, urmată de transformarea obiectului în sistemul de referință ecran 3D (fig. 10.1). Prin parametrizarea suprafețelor se asociază un punct de coordonate  $(s, t)$  în spațiul texturii fiecărui vârf al obiectului modelat (reprezentat în sistemul de referință de modelare). Transformarea completă de aplicație a texturii mai necesită, după parametrizarea suprafețelor, toate transformările geometrice obișnuite, de la sistemul de referință model (spațiul obiect) la sistemul de referință ecran 3D.

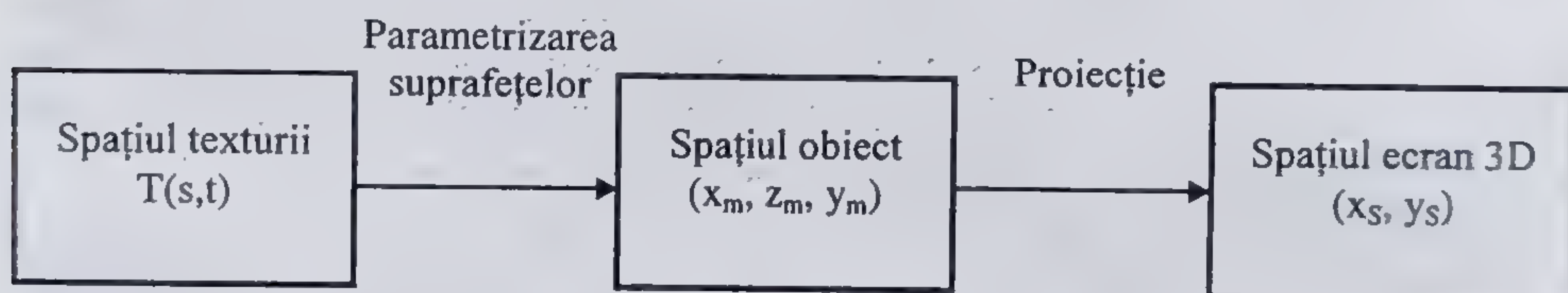


Fig. 10.1 Aplicația texturilor bidimensionale.

Dacă se compun cele două transformări (parametrizarea suprafeței și transformarea în spațiul ecran 3D) se obține o transformare de aplicație a texturii (*texture mapping*), care transformă puncte din spațiul texturii în spațiul ecran 3D. În fig. 10.2 (a), textura este o tablă de șah, definită în spațiul texturii. Această textură este aplicată unor suprafețe prin definirea coordonatelor fiecărui vârf al suprafeței în spațiul texturii, iar prin transformarea în spațiul ecran 3D și transformarea de rastru, se obține imaginea suprafețelor texturate (fig. 10.2).

Transformarea de parametrizare amplasează imaginea texturii pe un obiect prin asocierea unui punct  $(s, t)$  în spațiul texturii (parametrul de texturare), fiecărui punct  $(x_m, y_m, z_m)$  al obiectului. Modul cum poate fi colată o suprafață plană (textura) pe un obiect tridimensional nu are o soluție unică și nu poate fi realizat decât admitând diferite tipuri de erori și distorsiuni. În implementările practice au fost adoptate mai multe tehnici de aplicație ale texturilor. De exemplu, se poate atribui câte o textură fiecărei fețe a unui obiect (așa cum este în Planșa 7), unui grup de fețe ale obiectului, sau se poate proiecta aceeași textură pe toate fețele obiectului.



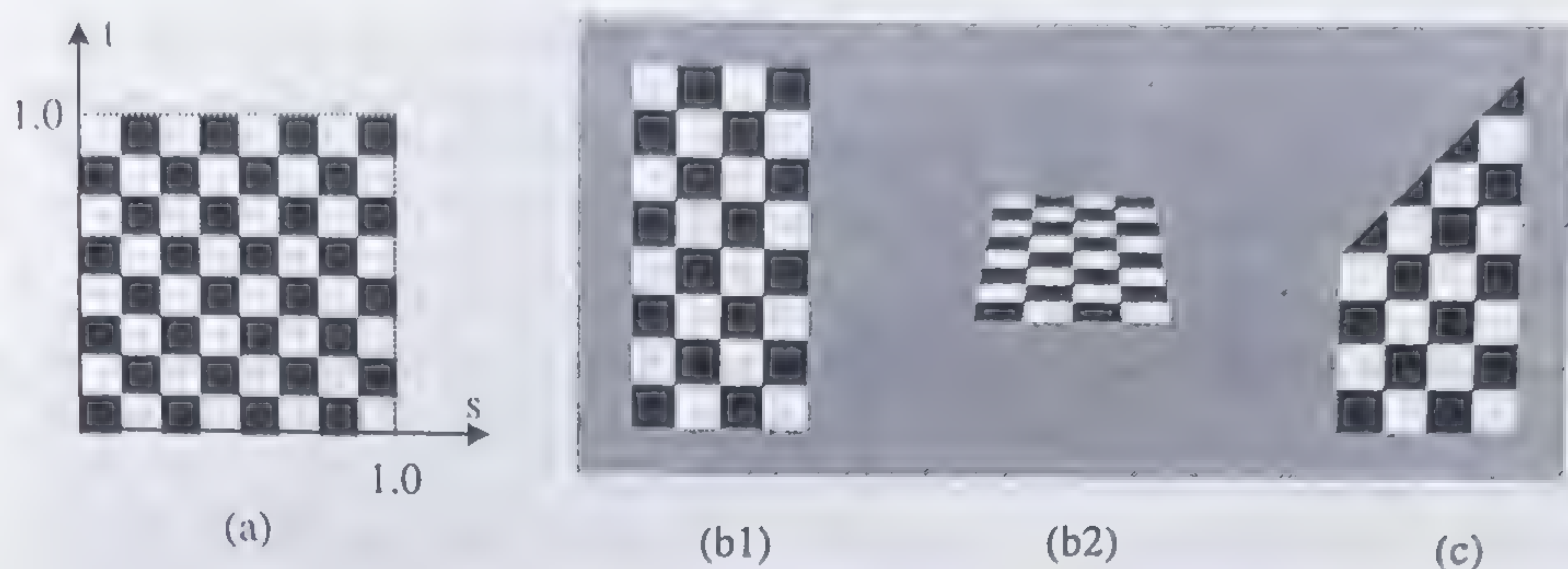


Fig. 10.2 Transformarea de aplicație a texturilor:  
(a) spațiul texturii; (b1), (b2), (c) imaginea suprafețelor texturate.

**Aplicația texturii pe o suprafață plană.** La acest mod de aplicație a texturii, se definesc coordonatele ( $s$ ,  $t$ ) în spațiul texturii pentru fiecare vârf al suprafeței. Aceste coordonate sunt calculate o singură dată, la modelarea obiectului, și sunt memorate împreună cu coordonatele vârfului și cu normala în vârf.

Dacă  $0 \leq s, t \leq 1$ , atunci imaginea texturii apare o singură dată (în întregime sau parțial) pe suprafața texturată. De exemplu, în fig. 10.2, prima suprafață texturată are coordonatele vârfurilor în spațiul texturii  $(0.0, 0.0)$ ,  $(0.5, 0.0)$ ,  $(0.5, 1.0)$ ,  $(0.0, 1.0)$ , și imaginile din fig. 10.2(b1) și (b2) pentru două puncte de observare diferite. Cea de a doua suprafață are coordonatele vârfurilor în spațiul texturii  $(0.5, 1.0)$ ,  $(1.0, 0.0)$ ,  $(1.0, 1.0)$ ,  $(0.5, 0.5)$  și imaginea din fig. 10.2(c).

Dacă coordonatele vârfurilor suprafeței în spațiul texturii depășesc pătratul cu latură egală cu 1 în care este definită imaginea de textură, atunci se poate ca textura să se repete pe suprafața texturată. De exemplu, în imaginea din Planșa 7a, fața verticală a cubului are coordonatele de textură ale vârfurilor  $(0.0, 0.0)$ ,  $(4.0, 0.0)$ ,  $(4.0, 4.0)$ ,  $(0.0, 4.0)$ . Textura (formată din două "cărămizi") se repetă de patru ori pe suprafață (se văd opt cărămizi).

Adresa texelilor corespunzători pixelilor unei primitive geometrice (obținuți prin transformarea de rastru) se calculează printr-o *transformare inversă transformării de aplicație a texturii*. Un pixel, definit ca un pătrat în spațiul ecran 3D, este transformat în spațiul obiect printr-o transformare inversă transformării model-ecran 3D, și apoi este transformat în spațiul texturii, printr-o transformare inversă transformării de parametrizare. Deoarece transformarea inversă transformării model-ecran 3D nu este liniară, unui pixel din imagine îi corespunde o suprafață în spațiul texturii care, în general, este un patrulater. Culoarea texelului corespunzător acestui patrulater este atribuită pixelului.

Calculul texelilor în spațiul texturii corespunzători pixelilor primitivelor geometrice este un proces de eșantionare și este posibil să apară un zgomot de spectru transpus (aliasing), atunci când frecvența de eșantionare este mai mică decât dublul frecvenței spațiale maxime a texturii. Prin analogie cu cele prezentate în capitolul precedent, frecvența spațială a texturii este raportul radian/dimensiune, iar frecvența spațială maximă a texturii este dată de inversa dimensiunii celului mai



mic detaliu din textură. Se observă că problemele de aliasing care apar prin eşantionarea imaginii la generarea primitivelor grafice au corespondență în aliasing la transformarea de texturare și, de aceea, trebuie să fie folosite tehnici de anti-aliasing, adică filtrarea texturilor. Aceste aspecte sunt prezentate în subcapitolul următor.

**Aplicația texturii pe o rețea de poligoane.** Pentru aplicația unei texturi pe o rețea de poligoane care modelează un obiect tridimensional, Bier și Sloan au dezvoltat o tehnică cunoscută sub numele de aplicația texturii în două etape (*two-part texture mapping*), folosind o suprafață intermediară între suprafața texturii și grila de poligoane [Bier86]. Suprafața intermediară este în general neplanară, dar posedă o funcție de atribuire analitică prin care textura bidimensională este aplicată cu ușurință pe această suprafață. După această aplicație, corespondența dintre un punct al obiectului tridimensional și un punct în spațiul texturii se transformă în corespondența între spațiul tridimensional obiect și spațiul tridimensional al suprafeței intermediare. Prețul care se plătește în această abordare este faptul că apare o dublă distorsiune, deoarece se efectuează două aplicații: de la textură la suprafața intermediară și de la suprafața intermediară la obiect.

Suprafața intermediară poate fi un plan, un cilindru, o sferă sau un cub. Principiul metodei de aplicație în două etape a texturii, pentru o suprafață intermediară oarecare, poate fi descris astfel:

- (a) Aplicația de la spațiul bidimensional al texturii la spațiul tridimensional al suprafeței intermediare, numită aplicația S:  $T(s, t) \rightarrow T'(x_i, y_i, z_i)$ .
- (b) Aplicația de la textura tridimensională la suprafața obiectului, numită aplicația O:  $T'(x_i, y_i, z_i) \rightarrow O(x_m, y_m, z_m)$ .

Pentru aplicația de la spațiul bidimensional al texturii la spațiul tridimensional al suprafeței intermediare (aplicația S) se folosește ecuația parametrică a suprafeței intermediare. De exemplu, o suprafață cilindrică este definită parametric prin valorile  $(\theta, h)$ . Pentru sferă se folosește definiția parametrică prin unghiurile azimut  $(\theta)$  și elevație  $(\phi)$ . Pentru suprafețele parametrice (Bézier, B-spline) se folosește direct ecuația parametrică a acestora.

Atunci când se folosește o sferă ca suprafață intermediară, apare problema distorsiunii aplicației din planul texturii pe suprafața sferei, care crește către poli sferei. Cubul folosit ca suprafață intermediară de texturare este echivalent din punct de vedere topologic cu sfera. O suprafață plană se poate aplica pe un cub prin pliere, fără să apară distorsiuni ale texturii, dar textura este decupată la limitele laturilor cubului.

Pentru aplicația de la textura tridimensională la suprafața obiectului (aplicația O) au fost propuse mai multe modalități de calcul al corespondenței dintre punctele pe suprafața intermediară de texturare și punctele obiectului. În fig 10.3 sunt prezentate patru dintre corespondențele folosite în sistemele grafice. Modalitățile de calcul al acestora sunt:



- (a) Intersecția dintre obiect și linia care unește punctul  $T'$  cu centrul obiectului.
- (b) Intersecția dintre obiect și raza de lumină care poenește din punctul de observare și are raza reflectată de suprafața obiectului astfel încât aceasta trece prin punctul  $T'$ .
- (c) Punctul de pe obiect a cărui normală trece prin punctul  $T'$ .
- (d) Intersecția dintre obiect și normala la suprafața intermediară care trece prin punctul  $T'$ .

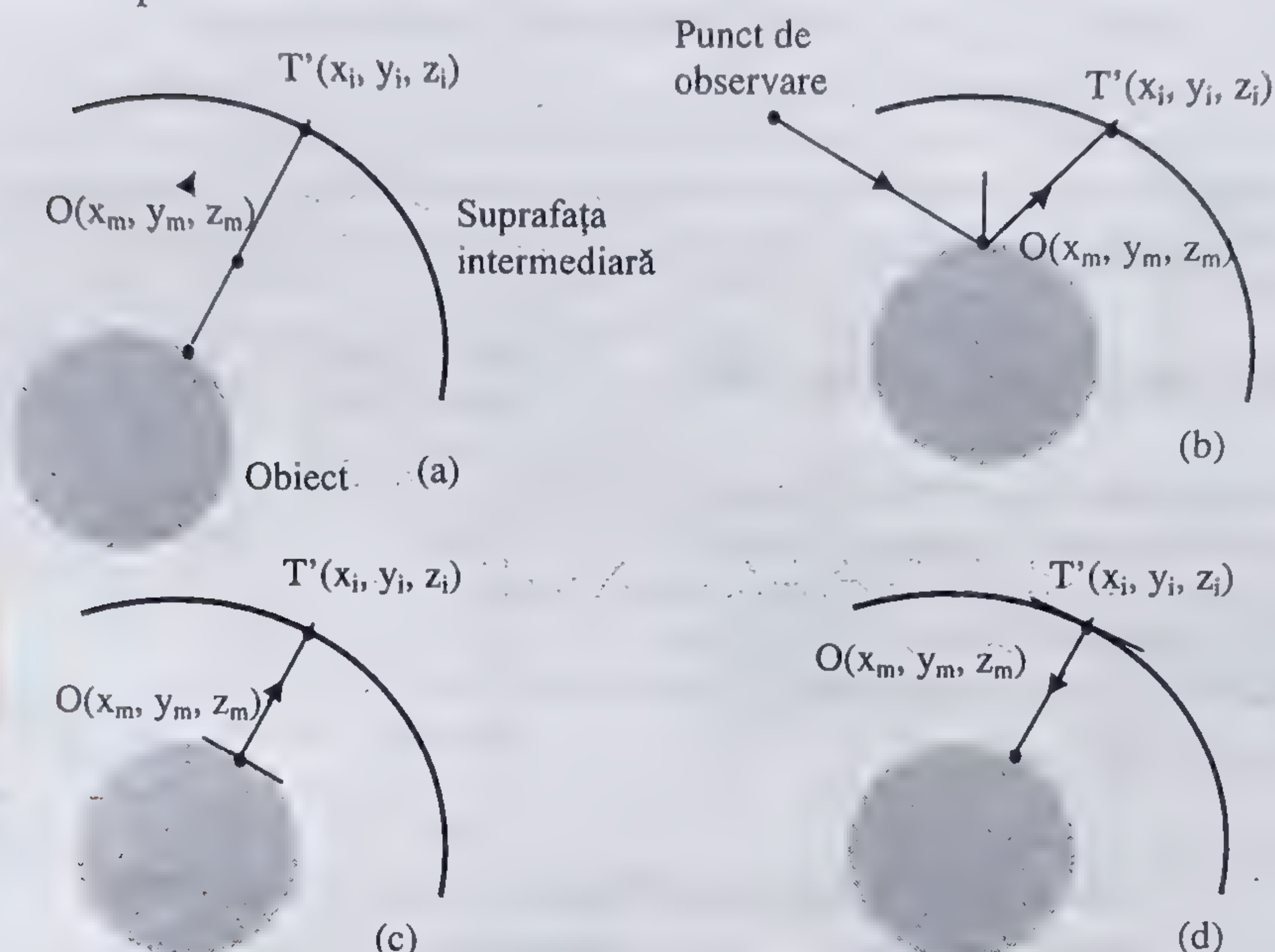


Fig.10.3 Aplicații de la suprafața intermediară de textură la obiect.

Împreună, cele două aplicații, fiecare cu câte patru posibilități, dau 16 combinații posibile de aplicații textură bidimensională-obiect.

Transformarea inversă aplicației texturii (numită transformare de texturare), se calculează pentru fiecare pixel din spațiul ecran 3D și constă din trei etape (fig. 10.4):

- (1) Transformarea inversă transformărilor modelare-observare-proiecție, prin care se trece de la coordonatele vârfurilor fiecărui pixel din spațiul ecran 3D, la coordonatele  $(x_m, y_m, z_m)$  a patru puncte pe suprafața obiectului (în spațiul obiect).
- (2) Transformarea inversă aplicației  $O$ , prin care se calculează coordonatele punctelor pe suprafața intermediară de texturare (cilindru, în figura de mai sus), corespunzătoare punctelor în spațiul obiect. Dacă se consideră cele patru puncte (vârfuri) ale pixelului, se obține imaginea acestuia pe suprafața intermediară de texturare (spațiul



texturii tridimensionale). Pentru suprafața intermediară cilindru, această transformare este:

$$(x_m, y_m, z_m) \rightarrow (\theta, h) = \left( \frac{1}{\tan(y_m/z_m)}, z_m \right) \quad (10.1)$$

- (3) Transformarea inversă aplicației  $S$ , prin care se calculează coordonatele  $(s, t)$  ale punctelor în spațiul texturii bidimensionale, corespunzătoare punctelor din spațiul texturii tridimensionale. Pentru suprafața intermediară cilindru, această transformare este:

$$(\theta, h) \rightarrow (s, t) = \left( \frac{r}{c}(\theta - \theta_0), \frac{1}{d}(h - h_0) \right) \quad (10.2)$$

unde  $c$  și  $d$  sunt factori de scalare, iar  $\theta_0$  și  $h_0$  sunt coordonatele cilindrice ale poziției texturii pe cilindrul de rază  $r$  și înălțime  $h$ .

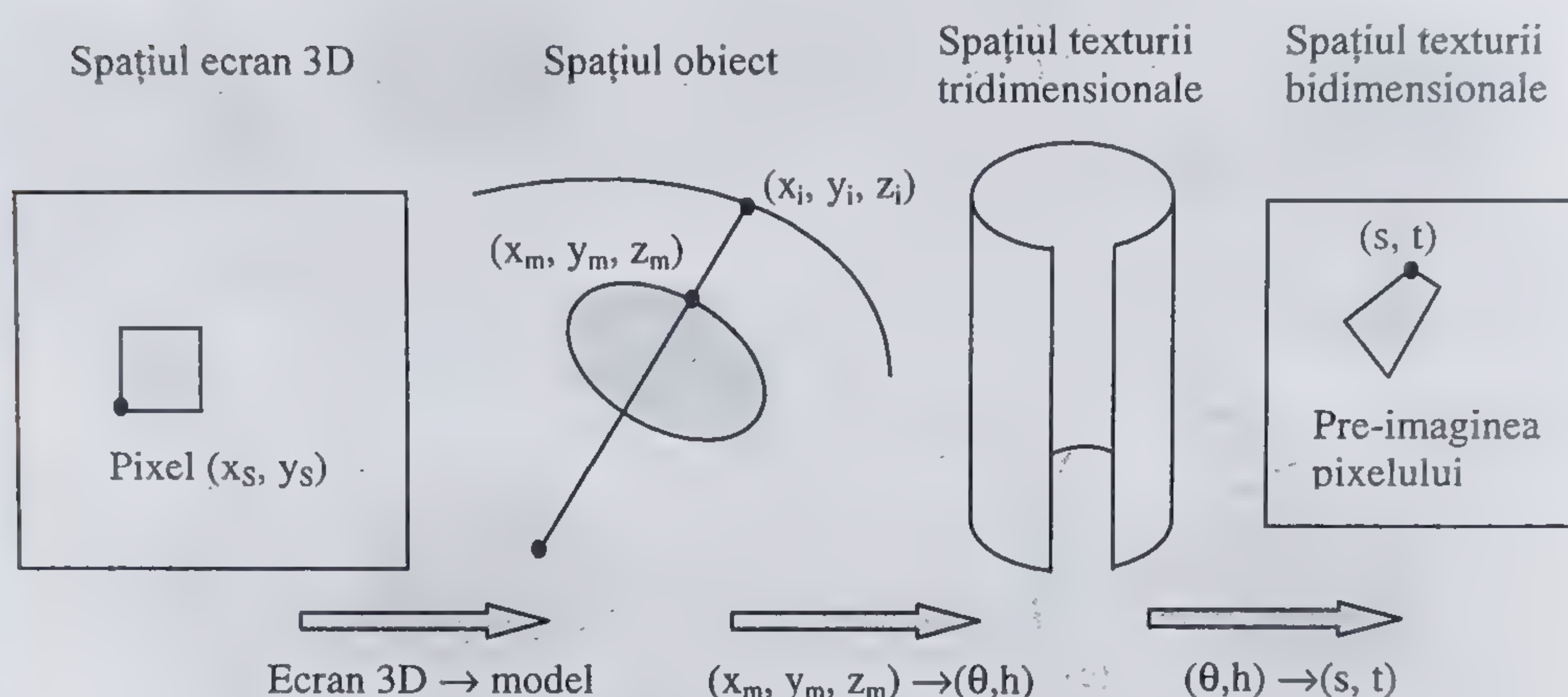


Fig. 10.4 Transformarea de texturare.

*Transformarea de parametrizare*, adică atribuirea coordonatelor de texturare fiecărui vârf al unui obiect se face, în general, în procesul de modelare a obiectelor. Modelul unui obiect va conține atunci toate informațiile necesare generării imaginii lui: coordonatele vârfurilor, normalele, materialul (culoarea) suprafeței, parametrii (coordoanatele) de texturare. Modelarea este un proces off-line, care se efectuează înainte de generarea interactivă a imaginii obiectelor și poate folosi metode laborioase de parametrizare, dat fiind că nu sunt impuse cerințe de execuție în timp real.

*Transformarea de texturare* se efectuează în cursul generării interactive a imaginilor. Toate aceste calcule, prin care se prelucrează texelii din imaginea de textură pentru calculul culorii pixelilor imaginii, trebuie executate pentru fiecare pixel al fiecărei suprafețe vizibile a obiectelor din scena virtuală, în fiecare cadru al imaginii. Din această cauză, transformarea de texturare trebuie executată cât mai eficient, iar răspuns în timp real nu se poate obține decât prin implementarea hardware a acesteia.



### 10.1.2 APLICAȚIA TEXTURILOR TRIDIMENSIONALE

Problema aplicației texturilor este mult mai direct rezolvată pentru texturile tridimensionale. Se poate imagina o textură tridimensională ca un câmp tridimensional continuu de valori în întreg domeniul în care este definit un obiect. Dacă se ignoră problemele de scalare între dimensiunea obiectului și dimensiunea texturii, atunci unui punct  $(x_m, y_m, z_m)$  de pe suprafața unui obiect îi corespunde un parametru de textură dat de aplicația identitate  $T(x_m, y_m, z_m)$ . Acest mod de aplicație poate fi imaginat ca o sculptură a obiectului într-un bloc solid de material, de unde și denumirea de texturare solidă (*solid texturing*) folosită pentru acest tip de aplicație a texturilor. Problema cea mai complicată în cazul texturilor tridimensionale este cantitatea de memorie imensă necesară pentru stocarea unui câmp tridimensional complet de valori. Pentru evitarea stocării unor volume foarte mari de date, implementarea texturilor tridimensionale se face prin generarea procedurală a câmpului de valori ale texturii. Coordonatele  $(x_m, y_m, z_m)$  sunt folosite ca argumente ale unei proceduri care definește câmpul texturii tridimensionale.

Avantajul principal al texturilor tridimensionale este acela al aplicației directe, fără distorsiuni, a texturii pe suprafața obiectelor. Dezavantajul acestui mod de texturare este că nu se poate obține procedural orice fel de imagine, mai ales imagini nerepetitive, de genul fotografiilor. De aceea, texturarea folosind texturi bidimensionale (numite și fototexturi) este procedeul cel mai flexibil și mai frecvent întâlnit în aplicațiile grafice.

## 10.2 TEHNICI DE ANTI-ALIASING ÎN TEXTURARE

Fenomenul de aliasing care apare în texturare are cauza în eșantionarea imaginii de textură. Transformarea din spațiul ecran 3D în spațiul texturii bidimensionale generează o suprafață în formă de patrulater corespunzătoare fiecărui pixel, care este numită *pre-imaginea pixelului*. Pentru simplificarea prezentării se presupune că pre-imaginea pixelului este un pătrat în spațiul texturii (fig. 10.5).

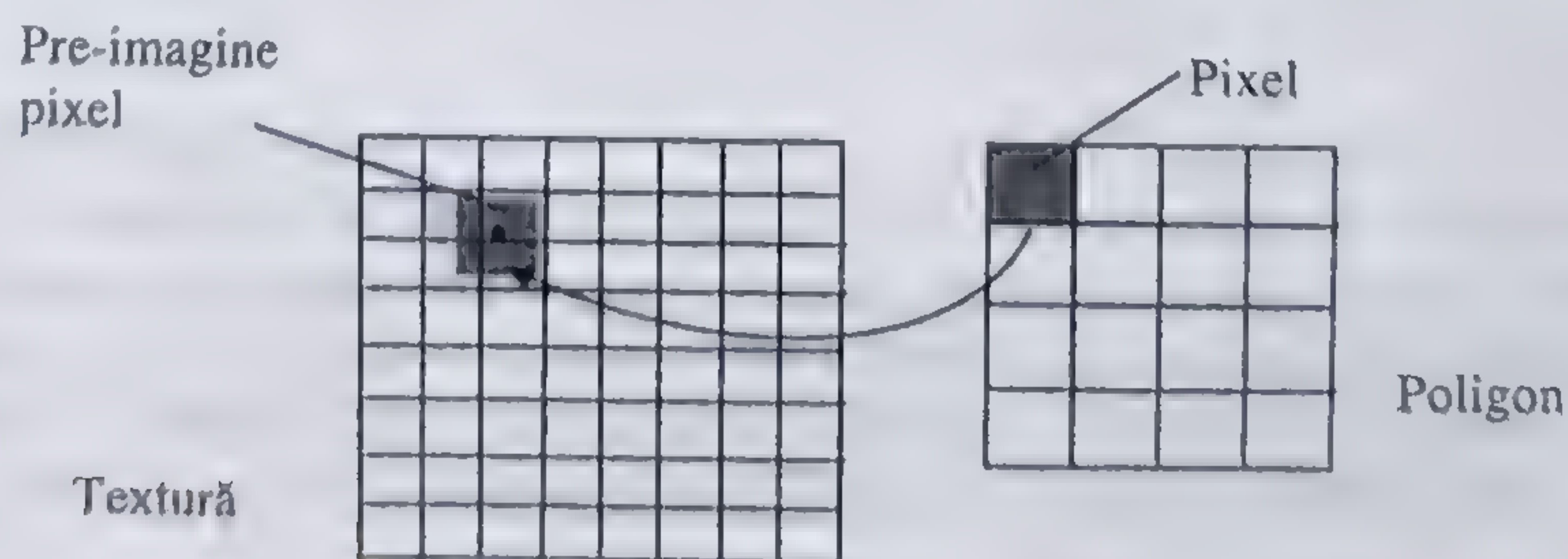


Fig. 10.5 Pre-imaginea unui pixel în spațiul texturii.



Pentru înțelegerea tehnicii de filtrare a texturii, se poate face o analogie directă cu tehnica de prefiltrare anti-aliasing, prezentată în capitolul 9. Imaginea texturii este eșantionată cu o frecvență de eșantionare dată de inversul dimensiunii pre-imaginei pixelului în planul texturii. Pentru eliminarea zgomotului de spectru transpus, se folosește un filtru trece-jos, aproximat cu fereastra Fourier pe o distanță inter-pre-imagine pixel, iar integrala de convoluție este o medie ponderată a tuturor texelilor acoperiți de pre-imaginea pixelului. În fig. 10.5, culoarea care se atribuie pixelului marcat se obține prin medierea ponderată a intensității culorilor celor patru texeli acoperiți parțial de pre-imaginea pixelului.

Dacă nu se efectuează filtrarea texturii, atunci se atribuie pixelului culoarea texelului cel mai apropiat de centrul pre-imaginei acestuia. Probele mele de aliasing care apar în această situație sunt foarte grave, mai ales pentru imaginile suprafețelor privite în perspectivă, în care variază mult dimensiunea pre-imaginei pixelilor.

Aria pre-imaginei unui pixel în planul texturii depinde de poziția suprafeței față de punctul și direcția de observare. Se presupune o suprafață pătrată cu parametrii texturii  $(0,0)$ ,  $(1,0)$ ,  $(1,1)$ ,  $(0,1)$ , perpendiculară pe direcția de observare, la o distanță  $D$  astfel că imaginea ei pe ecran este un pătrat de  $N \times N$  pixeli. Se mai presupune, de asemenea, că textura folosită are rezoluția de  $N \times N$  texeli. În această situație, aria pre-imaginei unui pixel în spațiul texturii este egală cu aria unui texel (fig. 10.6).

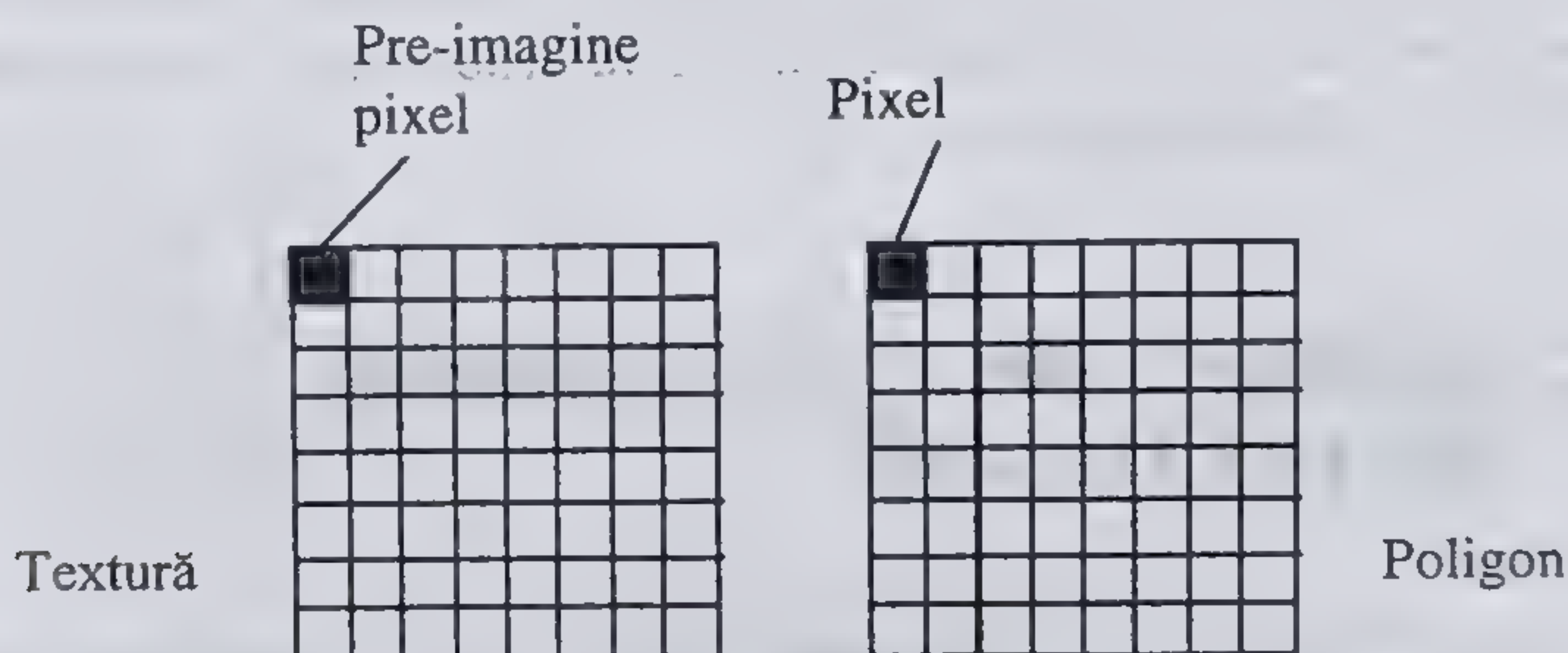


Fig. 10.6 Pixel cu pre-imagine cu aria egală cu aria unui texel.

Dacă suprafața se apropie de punctul de observare la distanță  $D/2$ , atunci imaginea suprafeței pe ecran este de  $2N \times 2N$  pixeli și imaginea celor  $2N \times 2N$  pixeli acoperă  $N \times N$  texeli, deci aria imaginii unui pixel este acum  $1/4$  din aria unui texel (fig. 10.7).

Cu cât suprafața se apropie mai mult, cu atât scade dimensiunea pre-imaginei pixelului în planul texturii și deci crește frecvența de eșantionare (inversul dimensiunii pre-imaginei pixelului) a texturii. În această situație nu apare zgomot de aliasing, dar imaginea nu devine mai bună, deoarece se observă discontinuitățile de trecere de la un texel la altul. Această situație se datorează faptului că textura nu este o imagine continuă, ci este reprezentată prin elemente discrete (texelii).



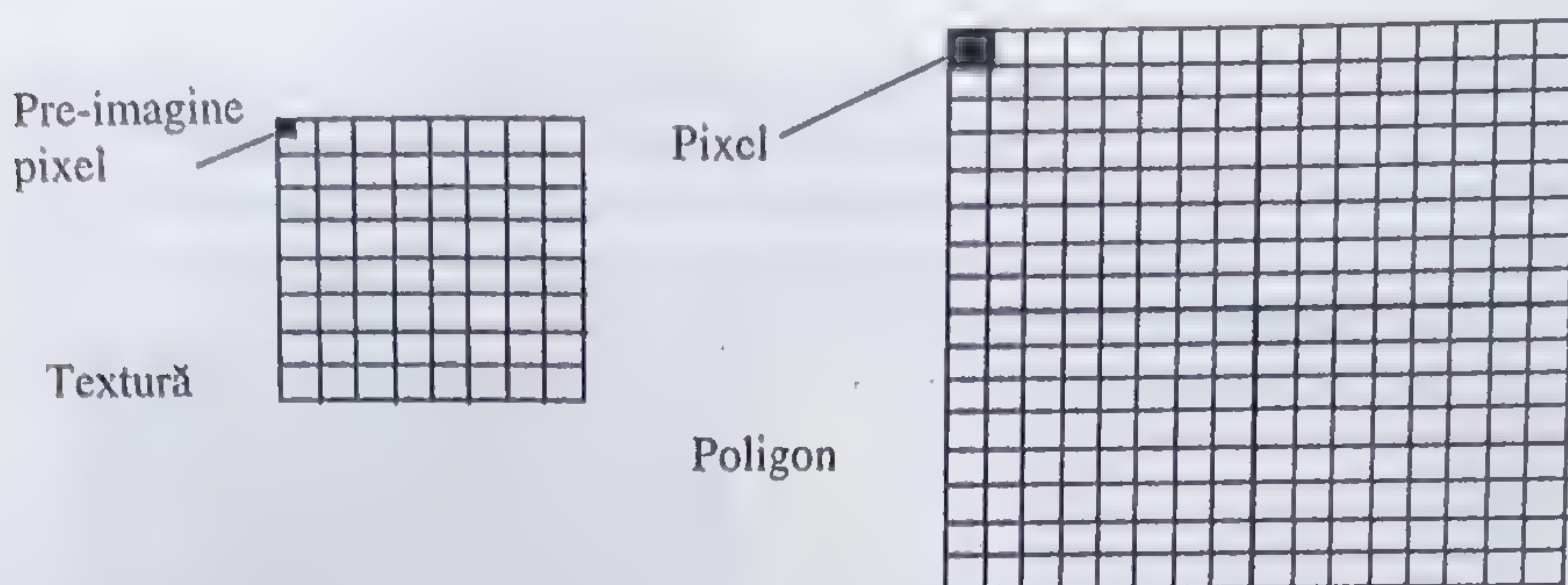


Fig. 10.7 Pixel cu pre-imagine cu arie mai mică decât aria texelului.

Dacă suprafața se depărtează de punctul de observare, de exemplu la distanța 2D, atunci suprafața are dimensiunea de  $(N/2) \times (N/2)$  pixeli pe ecran este și imaginea acestora acoperă  $N \times N$  texeli, deci aria pre-imaginii unui pixel este de patru ori mai mare decât aria unui texel (fig. 10.8).

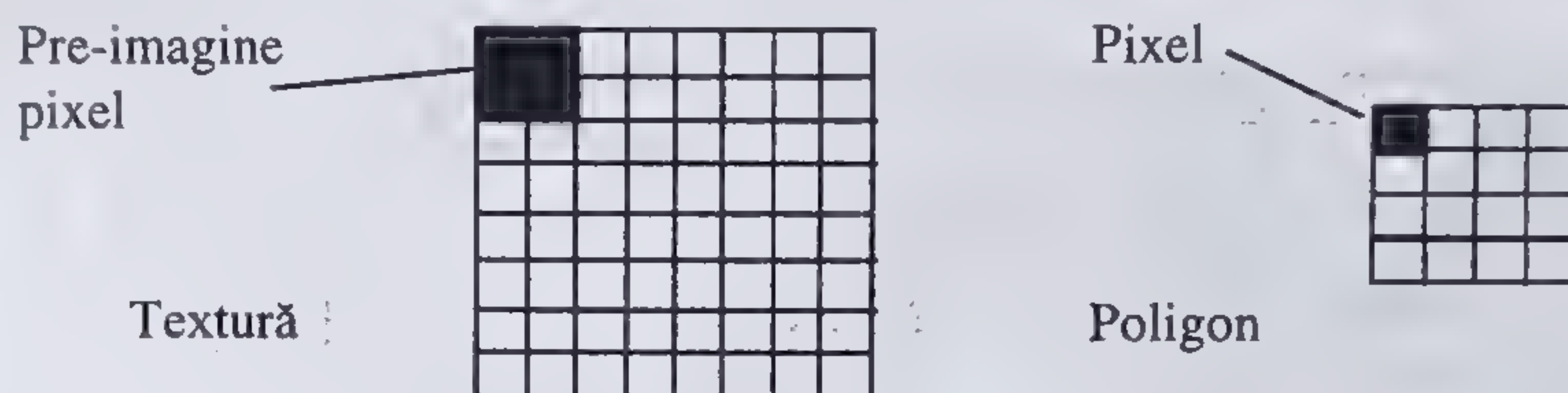


Fig. 10.8 Pixel cu pre-imagine cu aria mai mare decât aria unui texel.

Cu cât suprafața se depărtează mai mult, cu atât crește dimensiunea pre-imaginii pixelului în planul texturii și deci scade frecvența de eșantionare a texturii (inversul dimensiunii pre-imaginii pixelului). În această situație apare zgomot de aliasing, dacă nu se efectuează filtrarea imaginii texturii. Filtrarea se execută într-o fereastră Fourier pe dimensiunea pre-imaginii unui pixel în planul texturii, și se efectuează prin medierea ponderată a intensităților tuturor texelilor acoperiți de pre-imaginea pixelului.

Efectul de aliasing al texturii este deosebit de supărător, mai ales pentru texturi repetitive aplicate suprafețelor privite în perspectivă. Zgomotul în spectrul de joasă frecvență prin care se manifestă suprapunerea în spectrul frecvențelor spațiale înalte creează un aspect de "moaraj" al texturii nefiltrate. În Planșa 8 sunt reprezentate imaginile unei suprafețe fără filtrarea texturii (a) și cu filtrare (b).

**Filtre variabile în spațiu.** Pentru suprafețele observate în perspectivă, dimensiunea pre-imaginii pixelilor variază chiar în cuprinsul suprafeței, crescând cu creșterea distanței de observare. Acest lucru înseamnă că filtrarea texturii trebuie să fie realizată cu o fereastră de filtrare de dimensiune variabilă. Implementarea filtrelor variabile în spațiu (*space-variant filter*) a fost abordată de mai mulți cercetători, folosindu-se aproximarea cu un patrulater sau cu o elipsă a



ariei în spațiul texturii asupra căreia operează filtrul. Implementarea filtrelor variabile în spațiu are un timp de execuție ridicat și variabil în funcție de aria pre-imaginii pixelilor. Folosind această metodă, filtrarea texturilor de dimensiuni mari (cu număr mare de texeli) care se aplică unor suprafețe reprezentate într-o zonă mică pe ecran (număr mic de pixeli) este foarte costisitoare. Alte tehnici, cum este tehnica de prefiltrare a imaginii texturii, sunt mult mai eficiente și independente de aria suprafețelor texturare.

**Prefiltrarea texturilor.** Procedul de filtrare a texturii poate fi accelerat datorită faptului că imaginea care se filtrează (textura) este cunoscută și se pot construi imagini prefiltate ale acesteia. În loc să se efectueze medierea pe suprafața mai multor texeli atunci când pre-imaginea pixelului acoperă un număr mare de texeli, se poate folosi o imagine de textură prefiltrată, cu rezoluție mai scăzută, în care culoarea fiecărui nou texel reprezintă media culorilor texelilor din textura originală (fig. 10.9). Prin prefiltrarea texturii se reduce frecvența spațială maximă a texturii, în același raport cu reducerea frecvenței de eșantionare, dată de inversul dimensiunii pre-imaginii pixelilor.

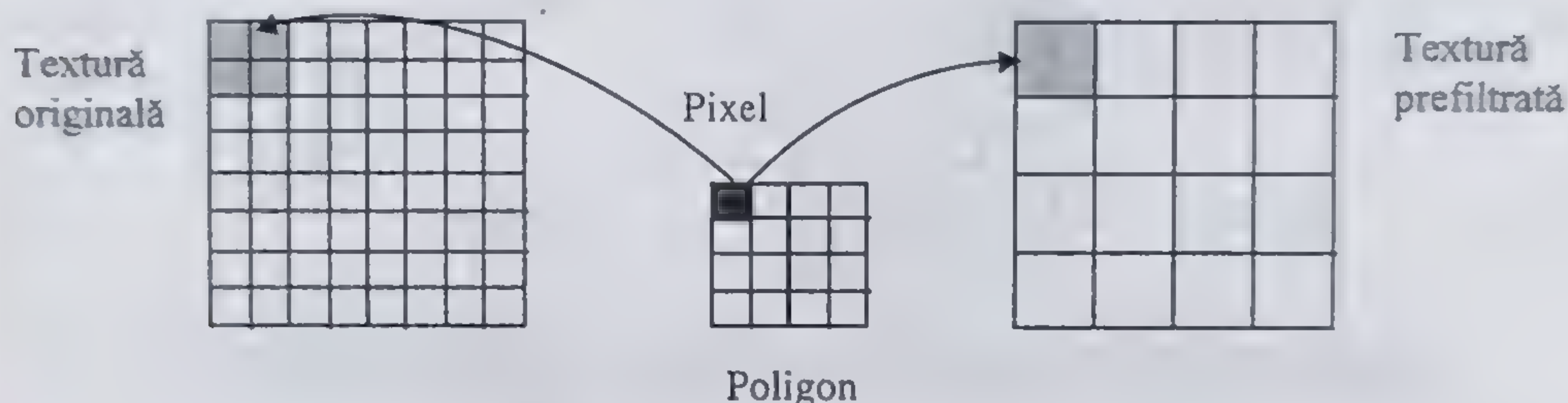


Fig. 10.9 Prefiltrarea texturilor.

O tehnică elegantă de prefiltrare a texturilor, care are o mare popularitate și numeroase implementări hardware în sistemele grafice, este tehnica numită *mip-map*, dezvoltată de Williams în 1983. În loc să fie folosită o singură imagine de textură, în tehnica *mip-map* se folosește o succesiune de imagini ale aceleiași texturi, toate derivate din imaginea originală, prin medierea culorilor și reducerea rezoluției. Fiecare imagine din succesiune are o rezoluție egală cu jumătatea rezoluției imaginii precedente și fiecare texel este media a patru texeli din imaginea precedentă. Termenul *mip* provine din expresia *multum in parvo* (multe lucruri într-un spațiu restrâns). Parametrul  $L$  este parametrul de selecție a nivelului de filtrare a texturii. Nivelul  $L = 0$  reprezintă imaginea originală a texturii (imaginea nefiltrată) cu rezoluția maximă. În general, în tehnica *mip-map* se folosesc rezoluții egale pe cele două coordonate  $s$  și  $t$ , cu valoare o putere a lui 2, iar imaginile prefiltate au rezoluții care scad prin înjumătățire până la valoarea  $1 \times 1$  (fig. 10.10).

Prin selecția nivelului de filtrare a texturii ( $L$ ), se alege imaginea cu rezoluția cea mai potrivită pentru dimensiunea pre-imaginii pixelului. Pentru evitarea discontinuităților între imagini la rezoluții diferite, se pot combina două



nivele de filtrare prin interpolare liniară. Selecția corectă a nivelului  $L$  este importantă. Dacă  $L$  este prea mare, imaginea apare estompată; dacă  $L$  este prea mic, atunci se observă zgomotul de aliasing. Nivelul  $L$  se alege astfel încât aria pre-imaginii pixelului să fie cât mai apropiată de aria texelului.

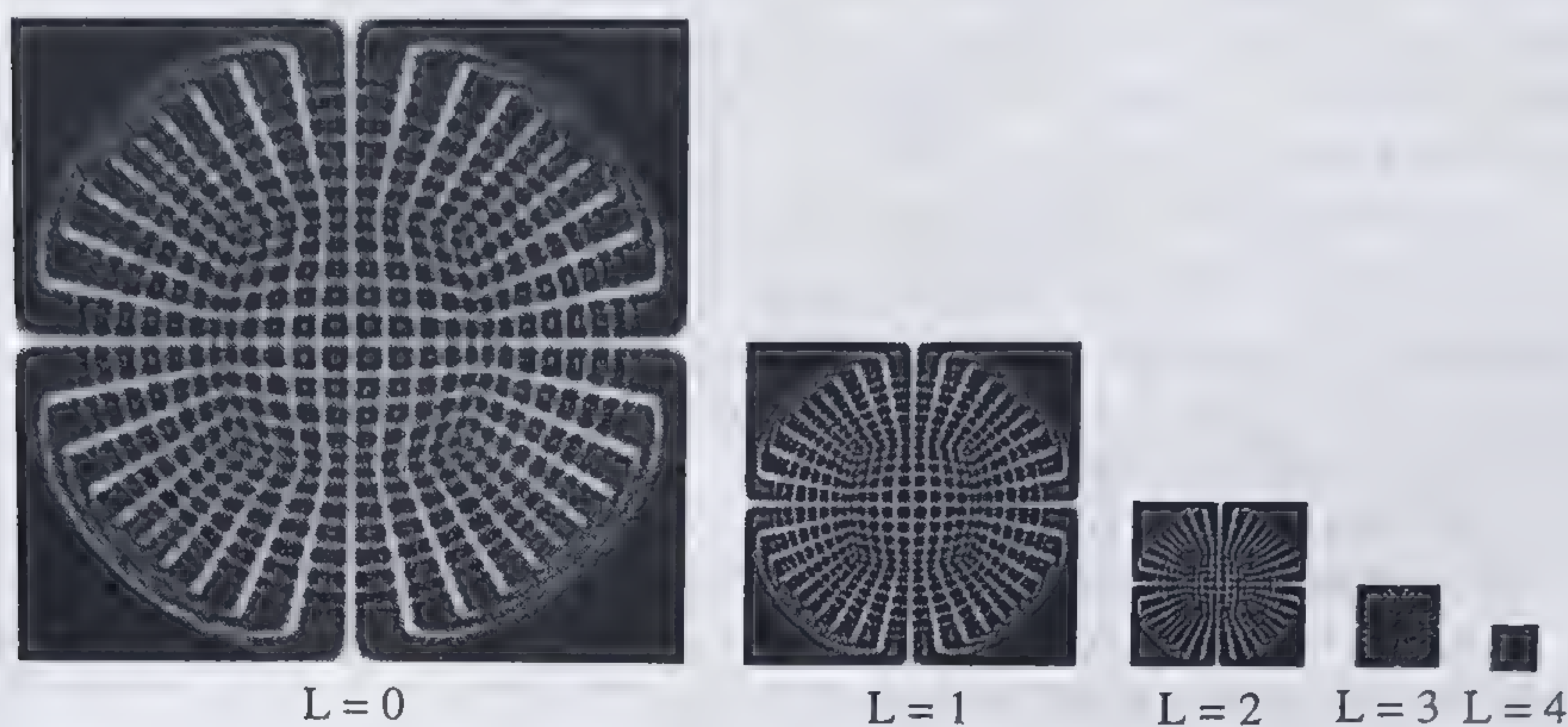


Fig. 10.10 Textură mip-map.

Detaliile de execuție ale prefiltrării folosind texturi *mip-map* depind de implementarea specifică, hardware sau software. În subcapitolul următor sunt prezentate funcțiile bibliotecii OpenGL prin care se pot programa aplicații grafice folosind texturarea suprafețelor.

### 10.3 FUNCȚII OPENGL DE TEXTURARE

Biblioteca OpenGL permite texturarea obiectelor, oferind suport pentru un număr foarte mare de posibilități de definire, aplicare și filtrare a texturilor. În lucrare vor fi prezentate numai o parte dintre aceste posibilități, ca exemplificare a modului de implementare a texturării și ca bază de pornire în crearea aplicațiilor grafice. Celelalte aspecte, mai de detaliu, se pot urmări în manualul de referință al bibliotecii, precum și în numeroasele exemple care sunt disponibile în OpenGL și GLUT.

Se pot defini texturi unidimensionale sau bidimensionale, cu rezoluții (număr de texeli) care se specifică la crearea imaginii texturii. Fiecare texel constă din una, două, trei sau patru componente, reprezentând valori de modulare sau cvadruple RGBA.

Texturarea poate fi validată sau invalidată prin apelul funcției `glEnable()` respectiv `glDisable()`, cu argument una din constantele simbolice `GLTEXTURE_1D` sau `GL_TEXTURE_2D` pentru texturi unidimensionale sau bidimensionale.



### 10.3.1 DEFINIREA TEXTURILOR

În OpenGL se pot defini mai multe texturi (în general, la inițializarea programului) și fiecare suprafață se texturează folosind una dintre acestea. Fiecare textură (obiect textură - *texture object*) are un nume unic în program și este definită printr-un set de date care permit aplicarea acesteia suprafețelor: adresa imaginii texturii, funcția de texturare (modulare, înlocuire, combinare), și proprietățile texturii (filtrări, mod de repetare, etc).

Posibilitatea de a manevra mai multe texturi în timpul execuției, fără să fie necesar încărcarea sau generarea imaginii texturii de fiecare dată, îmbunătățește performanțele de texturare. Deoarece OpenGL este o interfață de programare scrisă în limbajul C, obiectele textură sunt definite prin date și funcții separate, care trebuie specificate într-o anumită ordine:

- (a) Generarea numelor texturilor.
- (b) Crearea obiectelor textură și conectarea lor (*bind*) la numele texturilor.
- (c) Activarea unei texturi, pentru aplicarea acesteia primitivelor geometrice care urmează.

**Numele texturilor.** Numele texturilor sunt numere întregi fără semn (de tipul `GLuint`) care sunt memorate într-un vector transmis ca argument funcției:

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

Această funcție creează un vector de nume de texturi, unice în program, pe care le memorează în vectorul `textureNames`. Numele create nu sunt neapărat numere succesive. Funcția `glGenTextures()` creează numai numele texturilor și le marchează ca utilizate, dar obiectele textură se creează numai la conectarea (*bind*) acestora.

**Crearea texturilor.** Funcția `glBindTexture()` se folosește atât pentru crearea, cât și pentru utilizarea unei texturi. Prototipul ei este:

```
void glBindTexture(GLenum target, GLuint texName);
```

Parametrul `target` poate avea ca valoare una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D` pentru texturi unidimensionale, respectiv bidimensionale. Argumentul `texName` este numele unei texturi, generat de funcția `glGenTextures()` și memorat în vectorul de nume ale texturilor. Atunci când este apelată prima oară pentru un nume de textură, funcția `glBindTexture()` creează un nou obiect textură, cu toate datele referitoare la imaginea și proprietățile texturii implicite. După apelul funcției `glBindTexture()`, textura cu numele dat ca argument devine textură curentă și toate operațiile ulterioare, atât pentru definirea unor proprietăți ale texturii, cât și pentru aplicația texturii, folosesc textura curentă. Textura curentă se schimbă prin apelul unei noi funcții `glBindTexture()`.

**Crearea imaginii de textură.** Imaginea texturii este un tablou unidimensional sau bidimensional de texeli, fiecare texel având una, două, trei sau



patru componente. Semnificația componentelor texelilor se stabilește la crearea imaginii texturii prin definirea formatului intern al imaginii printr-un argument al uneia din funcțiile `glTexImage1D()`, `glTexImage2D()`.

Texturile unidimensionale au o utilizare restrânsă. Ele sunt folosite pentru texturarea în benzi, pentru care variația culorii are o singură direcție. În continuare se vor prezenta texturile bidimensionale, care sunt cel mai frecvent folosite. Funcția `glTexImage2D()` are următorul prototip:

```
void glTexImage2D(GLenum target, GLint level,
                  GLint internalFormat, GLsizei width,
                  GLsizei height, GLint border,
                  GLenum format, GLenum type, const GLvoid *pixels);
```

Parametrul `target` specifică crearea unei imagini de textură bidimensională (prin constanta simbolică `GL_TEXTURE_2D`) sau o interogare a capacității de memorare a unei imagini de textură (prin constanta simbolică `GL_PROXY_TEXTURE_2D`).

Parametrul `level` indică nivelul imaginii într-o succesiune de imagini prefiltrate de tipul mip-map. Nivelul 0 reprezintă imaginea originală, cu rezoluția maximă. Celelalte nivele se pot crea automat din imaginea originală (prin funcția `gluBuild2DMipmaps()`) sau pot fi create din tablouri de texeli, cu același format ca imaginea de nivel 0 a texturii.

Parametrul `internalFormat` indică tipul componentelor texelilor texturii, ca intensitate, luminanță sau R, G, B, A. Valoarea argumentului `internalFormat` poate fi un întreg între 1 și 4 sau una dintre treizeci și două de constante simbolice, dintre care cele mai frecvent folosite sunt: `GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`, `GL_RGB`, `GL_RGBA`. Aceste constante corespund valorilor 1, 2, 3, 4 ale argumentului. Aceste componente ale texelilor sunt combinate cu componentele culorii proprii (înainte de texturare) a pixelului pentru a crea culoarea finală a pixelului. Modul de combinare depinde de funcția de texturare, descrisă în continuare.

Parametrii `width` și `height` dau dimensiunea imaginii texturii, în număr de texeli, pe orizontală și pe verticală. Parametrul `border` reprezintă lățimea borderului imaginii de textură. Această caracteristică permite alipirea mai multor porțiuni de imagini într-o singură imagine de textură, iar borderul este folosit în operațiile de filtrare, în care sunt necesari texeli vecini texelului curent, pentru medierea culorilor acestora. Divizarea unei imagini de textură în mai multe părți este impusă în anumite implementări ale bibliotecii OpenGL, atunci când texturarea este implementată hardware și imaginea de textură se încarcă într-o memorie rapidă din sistemul grafic. Dacă imaginea texturii este prea mare pentru a fi stocată în întregime în memoria de textură, atunci se poate împărți în mai multe părți, care sunt încărcate și folosite succesiv. În această situație borderul permite filtrarea corectă la granițele părților componente ale imaginii.

Parametrii `format` și `type` se referă la formatul și tipul datelor imaginii de textură. Parametrul `format` poate fi una din constantele simbolice `GL_COLOR_INDEX`, `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`,



GL\_ALPHA, GL\_LUMINANCE, GL\_LUMINANCE\_ALPHA. Parametrul `type` poate fi GL\_BYTE, GL\_UNSIGNED\_BYTE, GL\_SHORT, GL\_FLOAT, GL\_INT, GL\_UNSIGNED\_INT, sau GL\_BITMAP. Ultimul parametru, `pixels`, conține datele imaginii de textură.

La apelul funcției `glTexImage2D()` se creează imaginea texturii curente (cea conectată prin funcția `glBindTexture()`) în formatul specificat.

**Funcții de texturare.** Modul în care se calculează culoarea unui pixel (sau fragment de pixel) se stabilește prin apelul uneia din funcțiile `glTexEnv#()`:

```
void glTexEnvf(GLenum target,
               GLenum pname, GLfloat param);
void glTexEnvf(GLenum target,
               GLenum pname, GLfloat param);
```

unde argumentele au următoarea semnificație:

- `target` specifică mediul de aplicare a texturii; trebuie să fie constanta simbolică `GL_TEXTURE_ENV`
- `pname` este numele simbolic al unui parametru de mediu de texturare; trebuie să aibă valoarea `GL_TEXTURE_ENV_MODE`
- `param` specifică funcția de texturare printr-o constantă simbolică care poate lua una din valorile `GL_MODULATE`, `GL_DECAL`, sau `GL_BLEND`.

Funcțiile de texturare stabilesc modul de calcul al culorii rezultante a pixelilor pe baza culorii texelilor din imaginea de textură și a culorii pixelilor primitivei geometrice (care se obțin din culoarea curentă sau din calcule de umbrire). În general, în modul `GL_DECAL` culoarea finală atribuită unui pixel este culoarea texturii; în modul `GL_MODULATE` se folosește culoarea texturii pentru modularea culorii fragmentului (pixelului); în modul `GL_BLEND` se combină culoarea texturii cu culoarea fragmentului. Această modalitate generală de atribuire a culorii depinde și de formatul intern al texturii, care se stabilește la crearea imaginii de textură.

**Atribuirea coordonatelor de texturare.** Coordonatele vârfurilor primitivelor geometrice în planul texturii (coordoanatele de texturare) se transmit prin funcțiile `glTexCoord#()`, care pot primi 1, 2, 3 sau 4 argumente de diferite tipuri, rezultând un număr mare de posibilități de apel. O parte dintre acestea sunt:

```
void glTexCoord1f(GLdouble s);
void glTexCoord2f(GLdouble s, GLdouble t);
void glTexCoord3f(GLdouble s, GLdouble t, GLdouble r);
void glTexCoord4f(GLdouble s, GLdouble t, GLdouble r,
                  GLdouble q);
void glTexCoord1f(GLfloat s);
void glTexCoord2f(GLfloat s, GLfloat t);
```



Coordonatele de texturare pot avea 1, 2 sau 3 valori pentru texturi unidimensionale, bidimensionale sau tridimensionale. În spațiul texturii, coordonatele sunt notate  $s$ ,  $t$ ,  $r$ , corespunzător coordonatelor  $x$ ,  $y$ ,  $z$  în spațiul obiect. Cea de-a patra componentă,  $q$ , este componenta de scală în reprezentarea texturii într-un sistem de coordonate omogen, asemănător sistemului de coordonate omogen folosit pentru reprezentarea punctelor în spațiu. Această coordonată este folosită dacă sunt necesare transformări ale texturii în coordonate omogene. În versiunea folosită în momentul de față a bibliotecii OpenGL (Versiunea 1.1), nu este încă implementată texturarea spațială și coordonata  $r$  nu este folosită, fiind prevăzută ca rezervă pentru utilizare ulterioară.

Dacă valorile coordonatelor de texturare în vârfurile primitivelor grafice sunt cuprinse în intervalul  $[0,1]$ , atunci textura este aplicată o singură dată pe suprafața respectivă. Dacă aceste valori depășesc intervalul  $[0,1]$ , atunci textura poate fi repetată pe suprafață sau limitată la intervalul  $[0,1]$ . Proprietatea unei texturi de a fi repetată sau limitată se stabilește prin apelul uneia din funcțiile:

```
void glTexParameterf(GLenum target, GLenum pname,
                     GLfloat param);
void glTexParameteri(GLenum target, GLenum pname,
                     GLint param);
```

În aceste funcții, parametrul `target` reprezintă tipul texturii și poate lua una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D`. Parametrul `pname` specifică numele simbolic al unei proprietăți a texturii, și poate lua una din constantele: `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAX_FILTER`, `GL_TEXTURE_WRAP_S`, `GL_TEXTURE_WRAP_T`. Primele două valori se referă la opțiunile de filtrare ale texturii și vor fi prezentate în paragraful următor. Următoarele valori setează proprietatea de repetare a texturii pentru coordonata  $s$ , respectiv  $t$ . În acest caz, parametrul `param` poate fi `GL_REPEAT` pentru repetarea texturii, sau `GL_CLAMP` pentru limitarea texturii la intervalul  $[0,1]$ . Mai multe aspecte privind definirea și folosirea texturilor în OpenGL vor fi detaliate în exemplele care urmează.

#### ■ Exemplul 10.1

În acest exemplu este prezentat programul prin care se aplică texturi în tablă de șah unor suprafețe în spațiu. Imaginea pe care o generează este o variantă a celei din fig. 10.2.

```
#include <GL/glut.h>

#define width 64
#define height 64

static GLubyte image4[height][width][4];
static GLubyte image8[height][width][4];
static GLuint texName[2];
```



```

void Makeimages(){
    int c;
    for (int i=0;i<height;i++){
        for (int j=0;j<width;j++){
            c = (((i&0x4)==0)^((j&0x4)==0)) *255;
            image4[i][j][0] = (GLubyte) c;
            image4[i][j][1] = (GLubyte) c;
            image4[i][j][2] = (GLubyte) c;
            image4[i][j][3] = (GLubyte) 255;
            c = (((i&0x8)==0)^((j&0x8)==0)) *255;
            image8[i][j][0] = (GLubyte) c;
            image8[i][j][1] = (GLubyte) c;
            image8[i][j][2] = (GLubyte) c;
            image8[i][j][3] = (GLubyte) 255;
        }
    }
}

void Init(void){
    glClearColor(0.6, 0.6, 0.6, 1.0);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_FLAT);

    /* TEXTURARE */
    Makeimages();
    glPixelStorei(GL_UNPACK_ALIGNMENT,1);
    glGenTextures(2, texName); // se creaza doua texturi

    /* Definirea primei texturi */
    glBindTexture(GL_TEXTURE_2D, texName[0]);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
        height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image4);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
        GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
        GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
        GL_NEAREST);

    /* Definirea celui de-a doua texturi */
    glBindTexture(GL_TEXTURE_2D, texName[1]);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
        height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image8);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_REPEAT);

```



```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                 GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                 GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST);
glEnable(GL_TEXTURE_2D);
}
void Display() {
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D, texName[1]);
    glPushMatrix();
    glTranslated(-2.0, 0.0, -8);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, -1.0, 0.0);
        glTexCoord2f(0.5, 0.0); glVertex3f( 0.5, -1.0, 0.0);
        glTexCoord2f(0.5, 1.0); glVertex3f( 0.5,  1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-0.5,  1.0, 0.0);
    glEnd();
    glPopMatrix();
    glPushMatrix();
    glTranslated(0.0, 0.0, -8);
    glRotated(-70, 1.0, 0.0, 0.0);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, -1.0, 0.0);
        glTexCoord2f(0.5, 0.0); glVertex3f( 0.5, -1.0, 0.0);
        glTexCoord2f(0.5, 1.0); glVertex3f( 0.5,  1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-0.5,  1.0, 0.0);
    glEnd();
    glPopMatrix();
    glBindTexture(GL_TEXTURE_2D, texName[0]);
    glPushMatrix();
    glTranslated(1.5, 0.0, -8);
    glBegin(GL_QUADS);
        glTexCoord2f(0.5, 0.0); glVertex3f(0.0, -1.0, 0.0);
        glTexCoord2f(1.0, 0.0); glVertex3f( 1.0, -1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0, 0.0);
        glTexCoord2f(0.5, 0.5); glVertex3f(-0.0,  0.0, 0.0);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h) {
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat) w/(GLfloat) h,
                  0.1, 4000.0);
    glMatrixMode(GL_MODELVIEW);
}

```



```

    glLoadIdentity();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Texture Check Image");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```

Imaginea captată din fereastra afișată la execuția acestui program este dată în fig. 10.11 și diferă de imaginea din fig. 10.2 prin faptul că se folosesc două texturi.

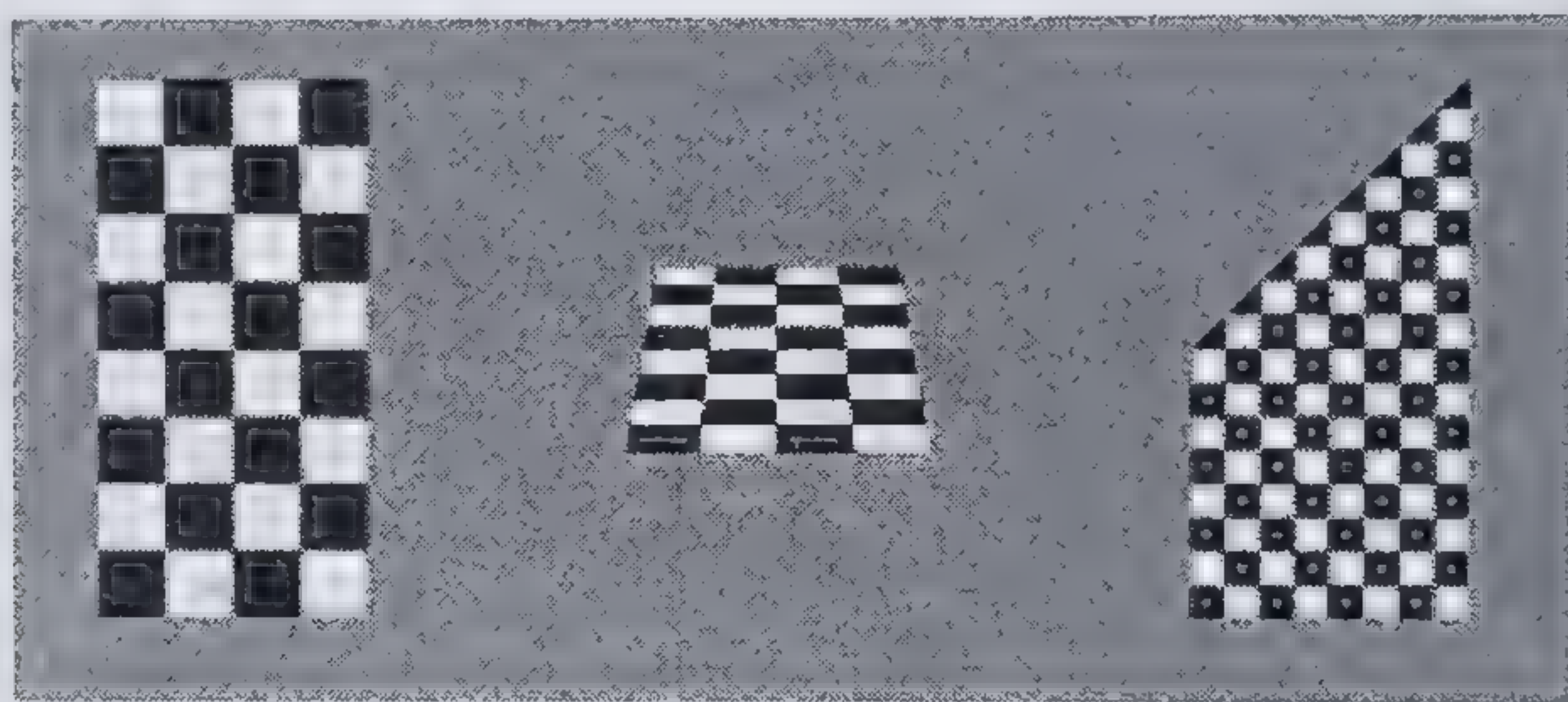


Fig. 10.11 Aplicația texturilor pe suprafețe plane.

În mod obișnuit, imaginile texturilor se citesc din fișiere, dar texturile simple în tablă de șah se pot genera prin program. Funcția `Makeimages()` creează două tablouri de  $64 \times 64$  locații, fiecare locație fiind un vector cu patru componente R, G, B, A. În primul tablou (`image4`) este creată o tablă cu  $16 \times 16$  pătrate alternante de culoare albă și neagră, fiecare pătrat de dimensiune  $4 \times 4$  texeli. În al doilea tablou (`image8`) este creată o tablă cu  $8 \times 8$  pătrate alternante de culoare albă și neagră, fiecare pătrat de dimensiune  $8 \times 8$  texeli.

Texturile sunt definite în funcția `Init()`. Mai întâi se creează numele a două texturi în vectorul `texName[2]`, prin funcția `glGenTextures()`. Pentru crearea și definirea proprietăților fiecărei texturi, se conectează mai întâi textura specificată prin numele ei (funcția `glBindTexture()`) și apoi se specifică proprietățile texturii.

În funcția `Display()` se utilizează texturile definite pentru texturarea unor primitive geometrice. Textura care se aplică este textura curentă, activată prin numele ei dat ca argument funcției de conectare `glBindTexture()`. Ca urmare,



primele două suprafețe sunt texturate cu textura cu numele `texName[1]`, iar a treia suprafață este texturată cu textura cu numele `texName[0]`, ceea ce se poate observa în imaginea din fig. 10.11. Imaginea din fig. 10.2 a fost generată cu același program, dar s-a activat numai textura cu numele `texName[1]`, adică textura formată din  $8 \times 8$  pătrate. Într-un bloc `glBegin()` - `glEnd()` se transmit vârfurile unei primitive geometrice; pentru fiecare vârf se definesc mai întâi coordonatele de texturare (cu `glTexCoord2f()`) și apoi coordonatele spațiale ale vârfului (cu `glVertex3f()`). Restul funcțiilor din program efectuează operațiile deja cunoscute, de transformări geometrice, setare culori de ștergere, etc.

### 10.3.2 STIVA MATRICELOR DE TEXTURARE

Cea de-a treia stivă de matrice de transformare OpenGL este stiva matricelor de texturare, care se activează prin comanda `glMatrixMode(GL_TEXTURE)`. Matricea din vârful acestei stivei se aplică automat coordonatelor de texturare. În mod implicit, aceasta este matricea identitate și coordonatele de texturare rămân nemodificate. Prin modificarea matricei din vârful stivei de texturare, se pot obține de efecte de îngustare sau lărgire a texturii (cu `glScale#()`), de rotație (cu `glRotate#()`) sau translație (cu `glTranslate#()`) a texturii pe suprafața obiectelor. Dat fiind că matricea de texturare este o matrice  $4 \times 4$ , se pot introduce și efecte de perspectivă în textură, dacă se folosesc coordonatele (s, t, r, q) de texturare.

#### ■ Exemplul 10.2

În acest exemplu se evidențiază folosirea stivei matricelor de texturare și modul de repetare sau limitare a texturii. Programul este o versiune ușor modificată a programului din exemplul precedent. În continuare sunt redată numai funcțiile care s-au modificat: `Init()`, `Display()` și `Reshape()`.

```
void Init(void) {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_FLAT);
    glClearColor(0.7, 0.7, 0.7, 1.0);
    MakeImages();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glGenTextures(2, texName); // se creaza doua texturi
    /* Definirea primei texturi */
    glBindTexture(GL_TEXTURE_2D, texName[0]);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
        height, GL_RGBA, GL_UNSIGNED_BYTE, image4);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_CLAMP);
```



```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                 GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MIN_FILTER, GL_NEAREST);
/* Definirea celei de-a doua texturi */
glBindTexture(GL_TEXTURE_2D, texName[1]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
             height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image8);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
          GL_DECAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                 GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                 GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glEnable(GL_TEXTURE_2D);
}

void Display(){
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D, texName[0]);
    glPushMatrix();
    glTranslated(-1.2, 0.0, -8);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 1.0);
        glTexCoord2f(1.0, 0.0); glVertex3f( 1.0, -1.0, 1.0);
        glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0, 1.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-1.0,  1.0, 1.0);
    glEnd();
    glPopMatrix();
    glBindTexture(GL_TEXTURE_2D, texName[1]);
    glPushMatrix();
    glTranslated(1.2, 0, -8);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 1.0);
        glTexCoord2f(1.0, 0.0); glVertex3f( 1.0, -1.0, 1.0);
        glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0, 1.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-1.0,  1.0, 1.0);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

```



```

gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 0.1, 4000);
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glScaled(2, 2, 2);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
)

```

Imaginea generată la execuția acestui program este prezentată în fig. 10.12.

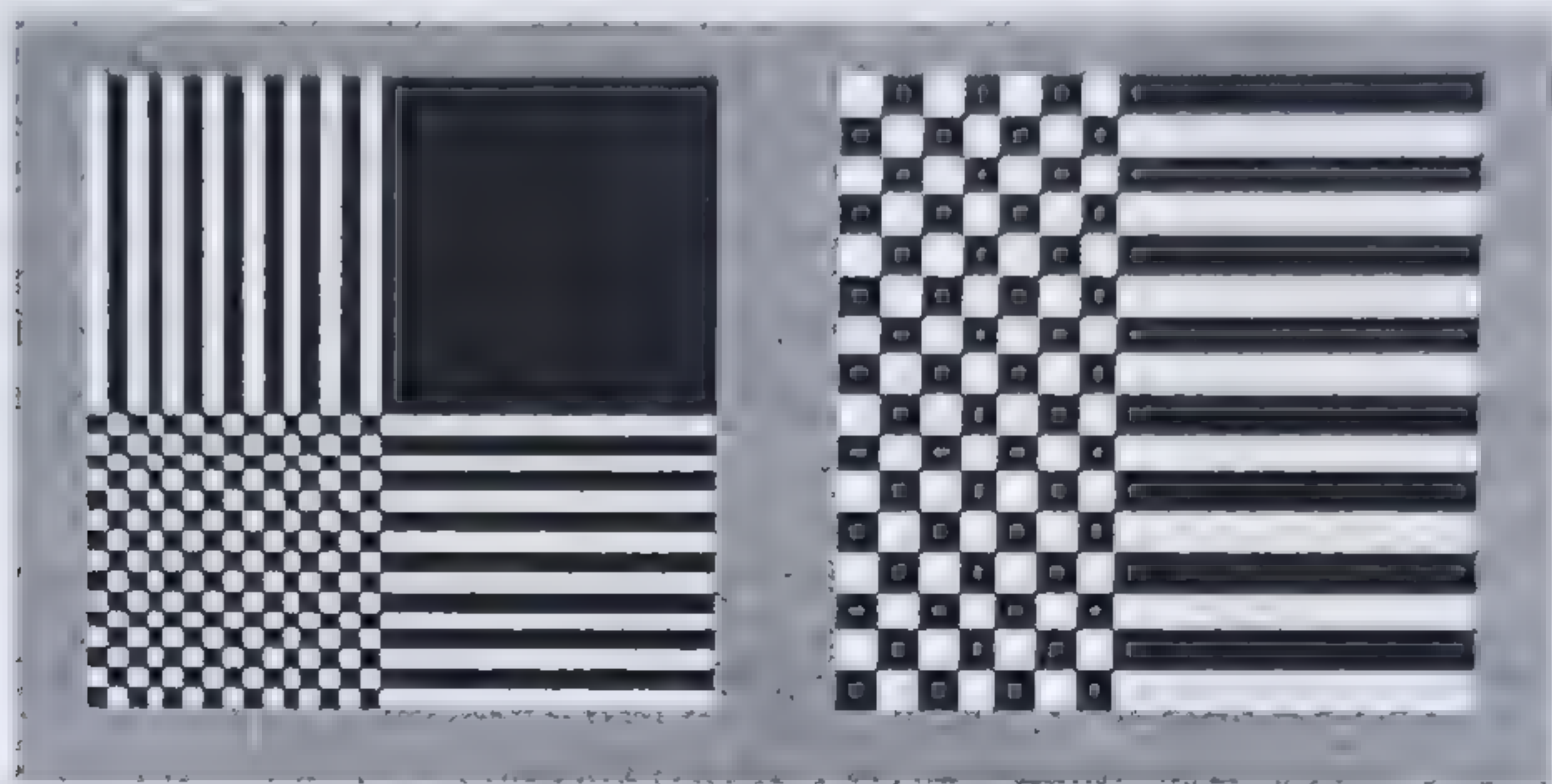


Fig. 10.12 Utilizarea stivei matricelor de texturare.  
Repetarea și limitarea texturilor.

În funcția `Reshape()` se selectează stiva matricelor de texturare (`glMatrixMode(GL_TEXTURE)`) și se introduce în vârful stivei o matrice de scalare (`glScaled(2.0, 2.0, 2.0)`). Această transformare se aplică coordonatelor de texturare ale celor două suprafețe desenate și deci aceste coordonate capătă valorile  $(0, 0)$ ,  $(0, 2)$ ,  $(2, 2)$ ,  $(2, 0)$ . La depășirea intervalului  $[0, 1]$ , textura este repetată sau limitată, în funcție de valoarea parametrilor `GL_TEXTURE_WRAP_S` și `GL_TEXTURE_WRAP_T` ai texturii. Prima textură este limitată pe ambele coordonate  $s$  și  $t$ , cea de-a doua textură este limitată pe coordonata  $s$  și repetată pe coordonata  $t$ . Imaginea din figura de mai sus evidențiază acest mod de aplicare a texturilor.

### 10.3.3 FILTRAREA TEXTURILOR

Tipul de filtrare care se aplică unei texturi este definit prin valorile a doi parametri, `GL_TEXTURE_MAG_FILTER` și `GL_TEXTURE_MIN_FILTER`, setați prin apelul funcției `glTexParameter#()`. Filtrarea de mărire (*magnification*, `GL_TEXTURE_MAG_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este egală sau mai mică decât dimensiunea texelului (fig. 10.13(a)). Filtrarea de micșorare (*minification*, `GL_TEXTURE_MIN_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este mai mare decât dimensiunea texelului (fig. 10.13(b)).



Dacă nu este definită imagine mip-map a texturii, cei doi parametri de filtrare `GL_TEXTURE_MAG_FILTER` și `GL_TEXTURE_MAG_FILTER` pot lua numai una din valorile `GL_NEAREST` sau `GL_LINEAR`. Valoarea `GL_NEAREST` înseamnă, de fapt, lipsa filtrării: se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului. Valoarea `GL_LINEAR` asigură filtrarea texturii prin calculul mediei ponderate a patru texeli cei mai apropiați de centrul pre-imaginii pixelului. Dacă este implementată software, filtrarea texturii (`GL_LINEAR`) este executată mai lent decât eșantionarea acesteia (`GL_NEAREST`).

În unele cazuri nu este simplu de decis dacă trebuie să se execute o filtrare de mărire sau o filtrare de micșorare. Patrulaterul prin care se reprezintă pre-imaginea pixelului poate să aibă dimensiunea într-o direcție mai mică decât dimensiunea texelului, iar în altă direcție mai mare decât dimensiunea texelului. În astfel de situații OpenGL selectează filtrul care dă cel mai bun rezultat posibil.

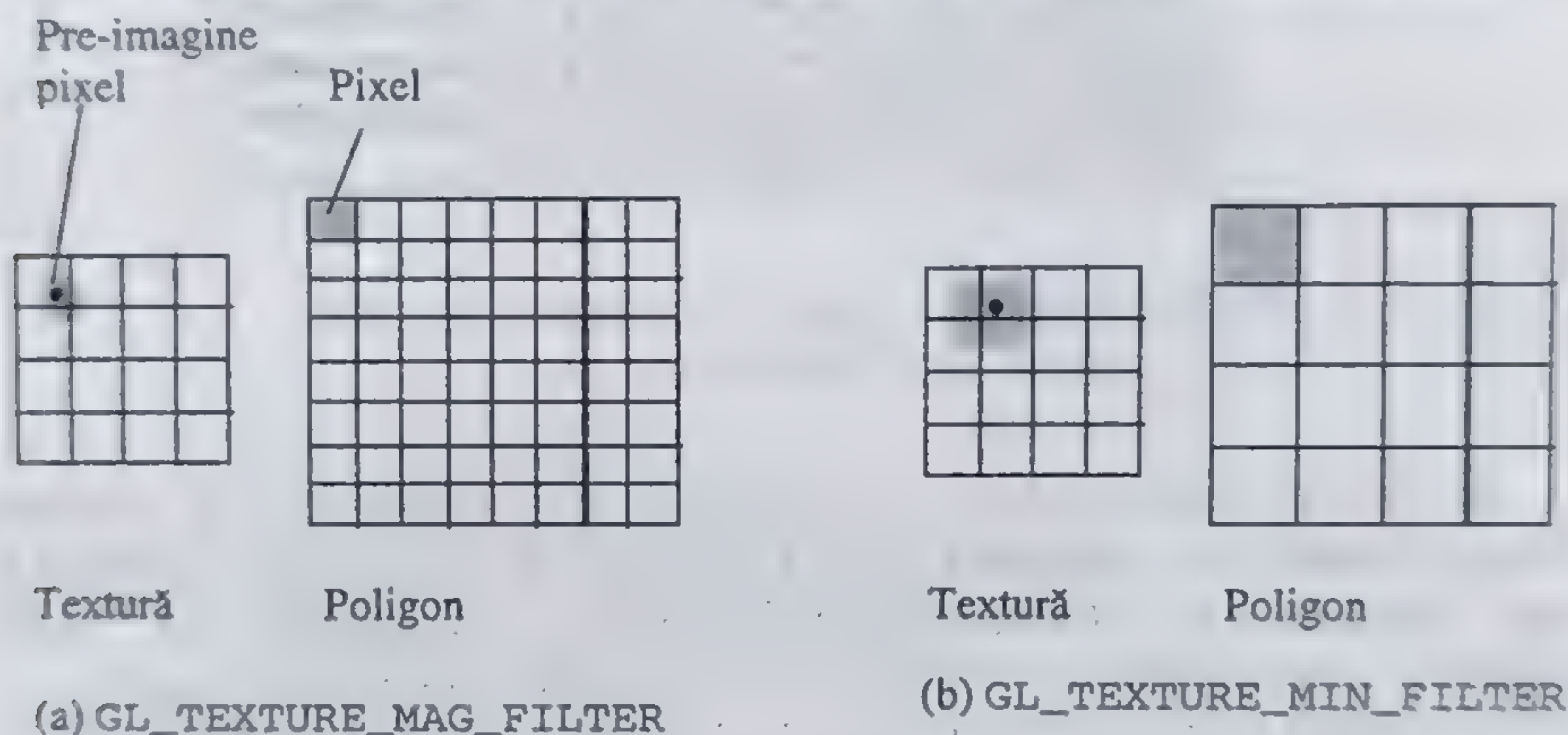


Fig. 10.13 (a) Filtrarea de mărire a texturii.  
(b) Filtrarea de micșorare a texturii.

Filtrarea de micșorare este accelerată prin tehnica de prefiltrare a texturilor folosind imagini de texturi mip-map. Secvența de imagini ale texturii mip-map poate fi generată prin apelul funcției `glTexImage2D()` pentru fiecare nivel, începând cu nivelul 0 (rezoluție maximă) până la ultimul nivel, cu rezoluție  $1 \times 1$ . Datele fiecărei imagini (tabloul bidimensional de texeli) se obțin din imaginea precedentă prin înlocuirea fiecărui grup de patru texeli cu un texel a cărui culoare este media culorilor celor patru texeli din imaginea precedentă.

Dacă s-a definit imaginea mip-map a texturii, atunci filtrarea de micșorare se poate realiza în mai multe moduri, depinzând de felul în care se selectează nivelul imaginii mip-map și de tipul de filtrare în imaginea mip-map selectată. Filtrul de micșorare (`GL_TEXTURE_MIN_FILTER`) este definit printr-una din următoarele constante simbolice:



- **GL\_NEAREST**: nu se aplică nici-o filtrare, se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului din imaginea de nivel 0 a texturii mip-map;
- **GL\_LINEAR**: se selectează imaginea de nivel 0 a texturii mip-map și se mediază ponderat patru texeli cei mai apropiați de centrul pre-imaginii pixelului;
- **GL\_NEAREST\_MIPMAP\_NEAREST**: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului și în această imagine se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului;
- **GL\_LINEAR\_MIPMAP\_NEAREST**: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului și în această imagine se mediază ponderat patru texeli cei mai apropiați de centrul pre-imaginii pixelului;
- **GL\_NEAREST\_MIPMAP\_LINEAR**: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul **GL\_NEAREST** (texelul cel mai apropiat de centrul pre-imaginii pixelului); acești texeli se mediază ponderat pentru obținerea valorii finale a culorii pixelului;
- **GL\_LINEAR\_MIPMAP\_LINEAR**: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul **GL\_LINEAR** (media ponderată a patru texeli cei mai apropiați de centrul pre-imaginii pixelului) și aceste valori se mediază ponderat pentru obținerea valorii finale a culorii pixelului.

Filtrarea **GL\_LINEAR\_MIPMAP\_LINEAR**, care se mai numește și filtrare triliniară, este cel mai eficient mod de eliminare a zgomotului de aliasing al texturii, dar și cel mai costisitor din punct de vedere al puterii de calcul. Chiar din simpla enumerare a operațiilor efectuate, se poate observa cantitatea de calcule extrem de mare necesară pentru fiecare pixel al imaginii generate. Filtrarea triliniară a texturilor generează imagini deosebit de realiste, dar este fie lentă, dacă este executată software, fie costisitoare, dacă este implementată hardware.

În exemplul următor se prezintă modul de programare a filtrării texturilor și rezultatele care se pot obține cu diferite tipuri de filtrări.

### ■ Exemplul 10.3

Programul prin care s-au obținut imaginile din Planșa 8 este următorul:

```
#include <gl/glaux.h>
#include <GL/glut.h>
```



```

AUX_RGBImageRec *image;
void Plane(){
    glBegin(GL_QUADS);
        glTexCoord2f(0.0,0.0);glVertex3f(-1.0, 1.0, 1.0);
        glTexCoord2f(1.0,0.0);glVertex3f( 1.0, 1.0, 1.0);
        glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,-1.0);
        glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,-1.0);
    glEnd();
}
void Init(void){
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    image = auxRGBImageLoad("../textures/floor.rgb");
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
        image->sizeY, GL_RGB, GL_UNSIGNED_BYTE, image->data);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
        GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D,
        GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER,
        GL_NEAREST);
        //GL_LINEAR_MIPMAP_LINEAR);
    glEnable(GL_TEXTURE_2D);
}
void Display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(0.0, -2.2, -4.0);
    Plane();
    glPopMatrix();
    glutSwapBuffers();
}
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (GLfloat)w/(GLfloat)h, 0.1, 1000);
    glMatrixMode(GL_TEXTURE);
    glLoadIdentity();
    glScaled(8,8,8);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

```



```

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|
                        GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Texturare");
    Init();
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}

```

În funcția `Reshape()` se încarcă matricea din vârful stivei matricelor de texturare cu o matrice de scalare care multiplică coordonatele de texturare cu factorul 8. În acest program se folosește o singură textură, care este în permanență textura curentă. Toate funcțiile de definire a parametrilor și de aplicare a texturii se referă la această unică textură și nu a mai fost necesar atribuirea unui nume și conectarea (*bind*) texturii pentru definire și activare.

Imaginea texturii se citește dintr-un fișier, folosind funcția `auxRGBImageLoad()` care aparține unei biblioteci auxiliare din OpenGL, `glaux.lib`, care nu a fost încă descrisă. Această bibliotecă auxiliară este o versiune mai veche a sistemului de dezvoltare GLUT, care a înlocuit-o pentru cea mai mare parte din funcțiile de creare a ferestrei de afișare OpenGL, tratare a evenimentelor din sistem sau generare a unor obiecte tridimensionale. Deoarece în versiunea 3.6 a bibliotecii GLUT în care au fost scrise programele, nu există o funcție de încărcare a imaginilor de textură, s-a apelat funcția din `glaux.lib`. Prin funcții ale bibliotecii `glaux.lib` se pot încărca imagini de textură în format RGB sau în format bitmap DIB (*Device Independent Bitmap*).

Imaginea citită din fișier este folosită pentru crearea secvenței mip-map de imagini prefiltrate ale texturii. Această operație se poate efectua prin funcția `gluBuild2DMipmaps()`.

După crearea imaginii mip-map a texturii, operațiile de definire și aplicare a texturii sunt asemănătoare cu cele prezentate în exemplele precedente. Setarea tipului de filtrare se face în funcția `Init()` prin funcția `glTexParameterf()`.

Pentru imaginea din Planșa 8 (a) nu se aplică nici o filtrare de micșorare prin apelul funcției:

```

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_NEAREST);

```

Pentru imaginea din Planșa 8 (b) se aplică modul de filtrare trilineară prin apelul funcției:

```

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

```



În Planșa 8 se observă diferența dintre imaginea cu textură filtrată și imaginea cu textură nefiltrată.

### 10.3.4 GENERAREA COORDONATELOR DE TEXTURARE

În general, coordonatele de texturare se generează la crearea modelelor obiectelor tridimensionale. Pentru obiectele modelate prin fețe poligonale, fiecare vârf se definește prin coordonate, normală și coordonate de texturare. Aceste date sunt transmise ca date ale primitivelor geometrice (în blocul `glBegin()` – `glEnd()`) și prelucrate corespunzător.

OpenGL este o bibliotecă de redare, nu de modelare a obiectelor tridimensionale, ea nu are funcții de generare și stocare a modelelor obiectelor care să conțină, printre alte informații, și coordonatele de texturare ale obiectelor. Pentru crearea modelelor se folosesc sisteme de dezvoltare specializate, iar comenzile de generare a coordonatelor de texturare din OpenGL sunt utile numai în anumite situații, de exemplu, pentru generarea texturilor în mișcare, care necesită coordonate de texturare calculate într-un plan a cărui poziție depinde de punctul de observare.

Funcțiile de generare a coordonatelor de texturare sunt `glTexGen#()` cu mai multe variante în funcție de tipul argumentelor:

```
void glTexGend(GLenum coord, GLenum pname,
               GLdouble param);
void glTexGenf(GLenum coord, GLenum pname,
               GLfloat param);
void glTexGeni(GLenum coord, GLenum pname, GLint param);
```

Primul parametru, `coord`, specifică coordonata de texturare care se generează și poate lua una din valorile constante: `GL_S`, `GL_T`, `GL_R` sau `GL_Q`, pentru coordonatele `s`, `t`, `r`, `q` respectiv. Parametrul `pname` este numele simbolic (`GL_TEXTURE_GEN_MODE`) al funcției de generare a coordonatelor de texturare, iar parametrul `param` poate lua una din valorile simbolice `GL_OBJECT_LINEAR`, `GL_EYE_LINEAR`, `GL_SPHERE_MAP`.

Pentru parametrii `GL_OBJECT_LINEAR` și `GL_EYE_LINEAR` funcția de generare a coordonatelor de texturare realizează o aplicație a texturii în două etape folosind ca suprafață intermediară un plan. În modul `GL_OBJECT_LINEAR` planul este specificat în sistemul de referință de modelare și se obține o textură fixă pe obiect. În modul `GL_EYE_LINEAR` planul intermediar este specificat în sistemul de referință de observare și se poate obține o textură dinamică, care se mișcă pe obiect.

Parametrul `GL_SPHERE_MAP` definește o funcție de generare a coordonatelor de texturare printr-o aplicație în două etape folosind ca suprafață intermediară o suprafață sferică. Aplicația sferică a texturii dă posibilitatea de a reprezenta obiecte din materiale care simulează materiale reale (de exemplu, lemn, marmură, rocă, etc).



Aplicația unei coordonate de texturare generate de OpenGL este efectivă numai dacă a fost validată prin apelul uneia dintre funcțiile `glEnable(GL_TEXTURE_GEN_S)` sau `glEnable(GL_TEXTURE_GEN_T)` pentru coordonata `s`, respectiv pentru coordonata `t`.

#### ■ Exemplul 10.4

În programul care urmează este exemplificată aplicația sferică a unei texturi bidimensionale.

```
#include <gl\glaux.h>
#include <gl\glut.h>

AUX_RGBImageRec *image;
void Cube() {
    // aceeași funcție ca în exemplul precedent,
    // cu deosebirea că laturile sunt egale;
}

void Init() {
    glClearColor(0.6, 0.6, 0.6, 1.0);
    image = auxRGBImageLoad("../textures/circles.rgb");
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
        image->sizeY, GL_RGB, GL_UNSIGNED_BYTE, image->data);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
        GL_SPHERE_MAP);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
        GL_SPHERE_MAP);
    glTexParameterf(GL_TEXTURE_2D,
        GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER,
        GL_LINEAR_MIPMAP_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
        GL_REPEAT);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
}

void Display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glPushMatrix();
    glTranslatef(-2.0, -2.5, -8);
    glRotatef(30, 1, 0, 0);
```



```
glRotatef(20,0,1,0);  
glutSolidTeapot(1.4);  
glPopMatrix();  
glDisable(GL_TEXTURE_GEN_S);  
glDisable(GL_TEXTURE_GEN_T);  
glPushMatrix();  
glTranslatef(2.0, -2.5, -8);  
glRotatef(30,1,0,0);  
glRotatef(20,0,1,0);  
Cube();  
glPopMatrix();  
glutSwapBuffers();  
}
```

Funcțiile `main()` și `Reshape()` ale programului sunt aceleași cu cele din exemplul 10.2 și nu au mai fost prezentate din nou. Imaginea care se obține la execuția acestui program este dată în Planșa 9(a).

În acest program se folosește o singură textură mip-map cu filtrare de mărire (`GL_LINEAR`) și de micșorare (`GL_LINEAR_MIPMAP_LINEAR`). Pentru ceainic (Planșa 9(a)) se validează generarea automată a coordonatelor de texturare cu aplicație sferică (`GL_SPHERE_MAP`). Pentru cub (Planșa 9(b)) se transmit coordonatele de texturare prin comenzile `glTexCoord2f()` și este invalidată generarea automată a coordonatelor de texturare (`glDisable(GL_TEXTURE_GEN_S)`, `glDisable(GL_TEXTURE_GEN_T)`).

Texturarea prin aplicație sferică creează efecte deosebit de interesante. Ea este folosită frecvent în ceea ce se numește *aplicație a mediului* (*environment mapping*) prin care se redă un obiect perfect reflectiv, în care se reflectă toate obiectele din mediul înconjurător.



## APLICAȚII ALE REALITĂȚII VIRTUALE

Orice aplicație de realitate virtuală se desfășoară într-o scenă virtuală compusă dintr-o colecție de modele de obiecte tridimensionale. În mod obișnuit, crearea scenelor virtuale (*modelarea*) și vizualizarea (*redarea*) acestora se desfășoară ca două procese separate, datorită cerințelor de timp de execuție foarte diferențiate.

*Modelarea scenelor virtuale* este un proces off-line prin care se creează colecția de modele ale obiectelor care compun scena virtuală. Procesul de modelare a obiectelor se desfășoară asistat de calculator, în diferite sisteme de proiectare interactivă, care pun la dispoziția proiectantului funcții de creare a modelelor pornind de la diferite date de intrare.

Pentru aplicațiile de realitate virtuală pot fi folosite atât sisteme de proiectare automată de uz general, cum sunt, de exemplu Autocad sau 3DStudio, sau sisteme de proiectare special concepute pentru crearea scenelor virtuale, numite sisteme de generare a bazelor de date (DBGS – *Data Base Generation Systems*). Astfel de sisteme, cum sunt sistemul Multigen, Corypheus, Sense8 și multe altele, oferă numeroase posibilități de modelare a obiectelor tridimensionale, a terenului, și a fenomenelor atmosferice (nori, ceață, etc.). Rezultatul operației de modelare îl reprezintă modelele obiectelor sau colecțiile de modele care compun o scenă virtuală, și acestea sunt memorate sub forma de *baze de date grafice*, alcătuite dintr-unul sau mai multe fișiere.

*Redarea scenelor virtuale* este un proces care se desfășoară interactiv, de cele mai multe ori în timp real, în cursul diferitelor experimente de realitate virtuală (simulatoare de antrenament, muzee virtuale, etc.), în sisteme grafice numite generatoare de imagine (*Image Generators – IG*). În generatoarele de imagine se re creează modelele obiectelor și al scenei virtuale, folosind ca date de intrare fișierele bazei de date grafice. Pentru redarea eficientă a imaginii, scena virtuală este organizată ierarhic, pe mai multe nivele de ierarhie, ceea ce permite selectarea rapidă a obiectelor, moștenirea transformărilor geometrice și a atributelor, etc. Această organizare ierarhică se implementează printr-un graf aciclic direcționat, numit *graful scenei* (*scene graph*). Nodurile grafului conțin modele ale obiectelor sau informații de redare (poziția obiectelor, materiale, lumini, etc.).



## 11.1 BAZE DE DATE GRAFICE

Bazele de date grafice sunt reprezentarea pe suport de memorie (fișiere) a modelelor obiectelor și a scenelor virtuale și conțin toate informațiile necesare pentru redarea scenelor.

Bazele de date obișnuite, utilizate pentru stocarea, actualizarea și regăsirea unor informații după un anumit criteriu, se bazează pe diferite modele de date, ca de exemplu modelul ierarhic, modelul rețea, modelul relațional, modelul orientării pe obiecte. În momentul de față, majoritatea sistemelor de baze de date sunt sisteme relaționale care asigură eficiență și fiabilitate în actualizarea și regăsirea informațiilor memorate.

În cazul bazelor de date grafice, criteriul principal de selectare a unui model de date este acela de a asigura redarea eficientă a scenei virtuale în cadrul aplicațiilor de timp real. Datorită reprezentării sub formă de graf a scenei virtuale în majoritatea generatoarelor de imagine, bazele de date grafice sunt, în majoritatea cazurilor, organizate într-un model ierarhic care folosește reprezentarea relațiilor între entități prin legături funcționale explicite de tipul părinte-fiu. Modelul ierarhic al bazelor de date grafice este cel mai adecvat modului de utilizare a datelor pe care acestea le conțin.

Există un număr mare de formate de reprezentare a bazelor de date grafice, mai mult sau mai puțin complete din punct de vedere al informațiilor de construire a grafului scenei virtuale. Proliferarea formatelor bazelor de date îngreunează portabilitatea programelor de aplicații grafice, care trebuie să prevadă posibilități de interpretare pentru cât mai multe formate.

În orice generator de imagine, modelul scenei are o structură bine precizată, care depinde de scopul aplicației, iar datele importate din diferite formate de baze de date sunt convertite la formatul intern al scenei. Un tip de format de bază de date reprezintă un anumit mod de descriere a entităților componente ale bazei de date și a relațiilor dintre ele. Cele mai cunoscute dintre formatele de baze de date grafice sunt prezentate în tabelul 11.1.

**Tabelul 11.1**  
**Formate de baze de date grafice**

Denumire	Descriere
3ds	Format binar, 3DStudio, firma Autodesk
dxf	Format ASCII AutoCad, firma Autodesk
obj	Format ASCII, firma Wavefront Technologies
phd	Format de descriere poliedre, firma Silicon Graphics
iv	Format ASCII Open Inventor, firma Silicon Graphics
dwb	Format binar/ASCII, firma Corypheus
flt14	Format binar Open Flight, firma Multigen



Formatele *3ds* și *dxf* sunt formate de export ale sistemelor de proiectare de uz general 3DStudio și AutoCad de la firma AutoDesk. Ele nu au fost concepute special pentru proiectarea bazelor de date grafice, astfel că nu pot avea decât o utilizare limitată. De exemplu, se pot folosi pentru proiectarea unor obiecte tridimensionale individuale, care sunt apoi înglobate în scene descrise prin formate mai complete.

Formatul *obj* este un format simplu care poate descrie obiecte tridimensionale reprezentate prin rețea de poligoane. Firma Wavefront Technology a plasat în domeniu public un număr de modele în format *obj*, care a devenit în felul acesta destul de cunoscut. O bogată librărie de modele produse de firma Viewpoint (peste 3500 de modele tridimensionale reprezentând avioane, mașini, animale, organe anatomice, etc.) sunt disponibile în formate *dxf*, *obj*, *flt*. Modelul avionului F-16 din fig. 2.16 este un model în formatul *obj*.

Formatele *dwb* și *flt* sunt formate speciale de descriere a bazelor de date grafice, dezvoltate împreună cu sistemele de generare a bazelor de date de companiile Corypheus și, respectiv, Multigen. Aceste formate conțin informații complete privind obiectele, amplasarea și gruparea lor, materiale, lumini, din care se poate crea graful scenei în generatorul de imagine.

Formatul *iv* (Open Inventor), dezvoltat de firma Silicon Graphics, este un superset al limbajului de modelare VRML 1.0 (*Virtual Reality Modeling Language*), care în 1997 a devenit standard sub numele de VRML 97. Limbajul VRML 97 permite crearea scenelor virtuale și accesarea lor prin internet. O scenă virtuală în limbajul VRML este un format de bază de date grafice, asemănător altor formate de baze de date. Denumirea de limbaj VRML provine din faptul că specificațiile VRML conțin reguli sintactice și semantice precise, care permit verificarea corectitudinii construcțiilor folosite. Acest limbaj este descris în subcapitolul următor.

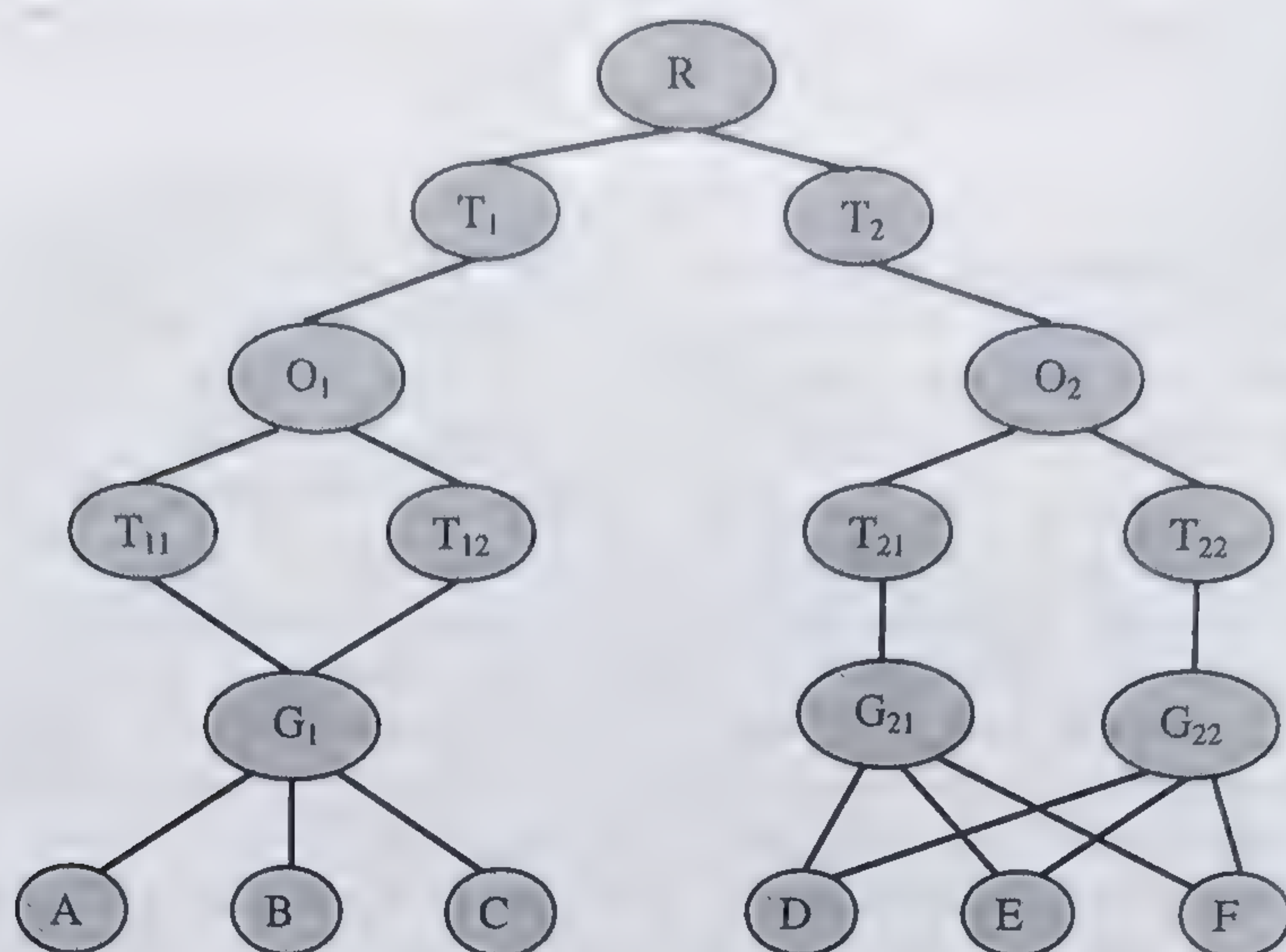
## 11.2 CREAREA ȘI REDAREA SCENELOR VIRTUALE

O scenă virtuală (*virtual scene*) este compusă dintr-o colecție de modele de obiecte tridimensionale, specificate prin forma și poziția lor, prin aspect (culoare, material, etc), și comportare (deplasare în spațiu, interacțiune). Diferite alte denumiri mai sunt utilizate cu același înțeles: lume virtuală (*virtual world*), mediu virtual (*virtual environment*). Construirea ierarhică a scenei permite organizarea și parcurgerea eficientă a acesteia. Scena ierarhică este compusă din noduri conectate prin arcuri direcționate. Un nod în scenă descrie o anumită entitate: un obiect tridimensional, o grupare de obiecte tridimensionale, o transformare geometrică, o textură, etc. Un arc al grafului scenei introduce descendenți (fii) ai unui nod, care moștenesc unele din atributele nodului părinte (fig. 11.1).

Construirea ierarhică a scenei permite reutilizarea obiectelor și a transformărilor geometrice. Un obiect complex se compune din gruparea mai



multor obiecte mai simple, dintre care unele se repetă, în poziții de instanțiere diferite. În felul acesta, un nod poate avea mai mult de un nod părinte.



**Fig. 11.1** Graful unui scene virtuale reprezentat printr-o ierarhie de noduri.

Nodurile grafului sunt de mai multe tipuri, dintre care cele mai importante sunt: nodurile de descriere a formei geometrice a obiectelor tridimensionale, nodurile de transformare și nodurile de grupare. Nodurile de descriere geometrică conțin modele ale unor obiecte elementare, reprezentate poligonal, parametric sau prin ecuația matematică într-un sistem de referință local (de modelare). Nodurile de transformare poziționează un nod fiu față de nodul părinte. Nodurile de grupare formează obiecte compuse din mai multe noduri fii.

În fig.11.1 nodurile A, B, C, D, E, F reprezintă obiecte tridimensionale. Nodul  $G_1$  grupează mai multe obiecte simple, formând un obiect compus care, la rândul lui, este instanțiat de două ori, în poziții definite prin intermediul nodurilor de transformare  $T_1$  și  $T_2$ , rezultând obiectul  $O_1$ , s.a.m.d. Nodul R este un nod de grupare și reprezintă rădăcina grafului scenei.

Graful scenei descrie relațiile ierarhice între noduri. Atributele unui obiect descris de un nod pot fi cele moștenite de la nodul părinte, pot fi modificate, sau redefinite în nodul fiu. Scena virtuală este definită în sistemul de referință universal, în care se poziționează și orientează punctul de observare în fiecare cadru al imaginii generate. Obiectele din nodurile frunză sunt reprezentate într-un sistem de referință local. Un nod de transformare conține o matrice de transformare  $M$  care definește poziția și orientarea sistemului de referință al nodului fiu în sistemul de referință al nodului părinte. Transformarea prin aplicarea matricei  $M$  punctelor obiectelor din nodul fiu le transformă din sistemul de referință al nodului fiu în sistemul de referință al nodului părinte.



Redarea imaginii unei scene virtuale reprezentate printr-un graf aciclic direcționat se realizează prin parcurgerea (*traversarea*) grafului scenei. Traversarea grafului scenei este o operație de parcurgere a grafului în adâncime (*depth first search*), care începe cu nodul rădăcină al grafului și parcurge toate drumurile în graf până la fiecare nod frunză. În acest fel, un nod poate fi traversat de mai multe ori, pe căi diferite de parcurgere. De exemplu, la parcurgerea grafului din fig. 11.1, nodul A este traversat de două ori, o dată pe calea  $RT_1O_1T_{11}G_1A$  și a doua oară pe calea  $RT_1O_1T_{12}G_1A$ ; la fel și nodurile B și C.

În reprezentarea ierarhică a obiectelor și a scenei, un obiect tridimensional este instanțiat (plasat în sistemul de referință universal) prin aplicarea unei secvențe de transformări de instanțiere, și anume toate transformările din nodurile de transformare de la rădăcină până la nodul frunză pe una din căile de parcurgere a grafului. Combinarea dintre operația de parcurgere în adâncime a grafului și operațiile de concatenare a matricelor de transformare prin intermediul stivelor de transformare permite reutilizarea matricelor calculate pe parcursul traversării prin salvarea lor în stivele de transformări și restaurarea atunci când sunt necesare.

Algoritmul de traversare a grafului scenei prezentat în continuare (în pseudocod asemănător limbajului C) specifică parcurgerea în adâncime a grafului scenei și operațiile care se execută la vizitarea fiecărui nod, în funcție de tipul acestuia. Se consideră un tip de date *Node*, care poate fi o clasă de obiecte (în programarea orientată pe obiecte) sau o structură (în abordarea procedurală) și o stivă de matrice de transformare de modelare, care admite operațiile de introducere (*glPushMatrix()*), de extragere (*glPopMatrix()*) și de concatenare (*glMultMatrix()*) a matricei curente de transformare *C* cu o nouă matrice *M*, cu depunerea rezultatului în matricea curentă ( $C = C M$ ). În mod obișnuit, aceasta este stiva OpenGL de modelare-vizualizare (stiva *MODELVIEW*). În această prezentare se consideră trei tipuri de noduri: noduri frunză (*SHAPE*), noduri grup (*GROUP*) și noduri de transformare (*TRANSFORM*).

```
void Traversal (Node* node){
    if (node->type == SHAPE){
        RedareObiect(folosind stiva curenta C)
    }
    else if (node->type == GROUP){
        for (i=0; i<nChilds; i++){
            glPushMatrix();
            Traversal(node->child[i]);
            glPopMatrix();
        }
    }
    else if (node->type == TRANSFORM){
        glMultMatrix(M);
        glPushMatrix();
        Traversal(node->child[i]);
        glPopMatrix();
    }
}
```



Notățiile care s-au folosit în acest algoritm sunt evidente. Tipul nodului ( $\text{node} \rightarrow \text{type}$ ) determină modul de prelucrare al acestuia. Un nod de grupare are  $n\text{children}$  noduri fii, numerotați  $\text{children}[i]$ . Un nod de transformare conține o matrice de transformare  $M$ . Traversarea începe prin inițializarea matricei curente de transformare cu matricea identitate ( $C = I$ ) și apelul funcției de traversare pentru nodul rădăcină al scenei ( $\text{Traversal}(R)$ ).

Matricele de instanțiere se introduc în stivă în ordinea inversă aplicării lor asupra obiectului, așa cum este necesar în operația de acumulare prin postmultiplicare. De exemplu, pentru prima instanță a obiectului  $A$ , se execută operațiile:

$$C = I$$

$$C = C M_1 = M_1$$

$$C = C M_{12} = M_1 M_{12}$$

Un punct oarecare  $P$  al obiectului  $A$  este instanțiat prin transformarea:

$$P' = CP = M_1 M_{12} P$$

Pentru o altă instanțiere a nodului  $A$ , matricea intermediară  $M_1$  este salvată înainte de prelucrarea fiecăruia dintre fiii nodului  $O_1$ . Poate este mai puțin evidentă avantajul metodei în acest exemplu foarte mic, dar pentru scene complexe, cu multe transformări succesive, economia de calcule de produse de matrice este considerabilă.

Acesta este modul principal de organizare și traversare a grafului unei scene virtuale. Construirea unui astfel de graf se poate face fie direct, prin program (pentru mici exemple cu scene de dimensiuni reduse) sau, în forma cea mai generală, prin parcurgerea unei baze de date descrise într-un limbaj (format) care să permită identificarea construcției ierarhice a scenei: tipuri de noduri, atributele acestora, relațiile ierarhice între noduri, etc.

### 11.2.1 CREAREA GRAFULUI SCENEI

În implementările actuale se utilizează frecvent programarea orientată pe obiecte pentru reprezentarea scenei virtuale, abordare care permite crearea unui program eficient, ușor de exploatat și de întreținut.

În continuare este prezentată o astfel de metodă de construire a grafului scenei, ca un proiect simplu și ușor de urmărit. Soluția adoptată pentru acest proiect (care va fi numit CSV – *Crearea Scenelor Virtuale*) urmărește modul de definire și creare a grafului scenei folosit în limbajul VRML și în mai multe din sistemele de creare a scenelor virtuale disponibile în momentul de față. Exemple de astfel de sisteme sunt: *Performer* (de la firma Silicon Graphics), *WorldUp* (de la firma Sense8), *dVS* (de la firma Division) *EasyScene* (de la firma Corypheus), *Multigen* (de la firma Software Systems, numită ulterior Multigen). Denumirile date claselor de obiecte în proiectul CSV sunt asemănătoare celor folosite în astfel de sisteme (*Node*, *Shape*, etc.). S-a considerat că folosirea unor denumiri în limba română ar



îngreuna remarcarea similitudinor obiectelor proiectului cu cele din sistemele cele mai folosite de creare și redare a scenelor virtuale.

Pentru descrierea orientată pe obiecte a grafului scenei virtuale se creează o ierarhie de clase, toate derivate dintr-o clasă de bază unică, numită clasa `Object`. Folosirea unei clase de bază unice pentru toate clasele permite definirea unui număr redus de clase pentru descriere diferitelor tipuri de liste liniare necesare pentru reprezentarea listelor de coordonate, de fețe, etc.

### 11.2.1.1 Vectorii de date

Clasa `ObArray` conține un vector de pointeri la clasa `Object`, alocat dinamic în memoria liberă. Reprezentarea prin vector a listelor de entități ale modelului (liste de coordonate de vârfuri, liste de fețe, etc.) este cea mai eficientă reprezentare pentru operația de prelucrare a listelor liniare, datorită accesului direct la elementele vectorului. Vectorul definit de clasa `ObArray` este vector cu auto-creștere dinamică, adică se realocă cu o dimensiune mai mare dacă se depășește dimensiunea alocată la un moment dat. Chiar dacă operația de realocare consumă timp de execuție ridicat, acest lucru nu este supărător deoarece vectorii se creează o singură dată, la crearea grafului scenei, înainte de experiența interactivă în realitatea virtuală.

```
class Object{
public:
    Object() {};
    ~Object() {};
};
class ObArray : public Object {
    int size;           // dimensiune curenta
    int grows;          // interval de creștere a dimens.
    int dimens;         // dimensiune alocata
    Object **p;         // vector de pointeri la Object
public:
    ObArray() {          // constructor implicit
        size = 0;
        grows = 4;
        dimens = grows;
        p = (Object **)new BYTE[grows * sizeof(Object*)];
    }
    ~ObArray();
    int Add(Object* x);
    int InsertAt(int i, Object *x);
    int RemoveAt(int i);
    .....
};
```

În mod asemănător se definește clasa `IntArray`, pentru vectori de numere întregi și clasa `DoubleArray`, pentru vectori de numere în virgulă flotantă dublă precizie.



```

class IntArray {
    int size;
    int grows;
    int dimens;
    int *p;
public:
    IntArray() {
        size = 0;
        grows = 4;
        dimens = grows;
        p = new int[grows];
    }
    ~IntArray() { delete p;}
    int GetSize() {return size;}
    int Add(int x);
    int GetAt(int i){return p[i];}
    int RemoveAt(int i);
    int& operator[](int i){return p[i];}
};

```

Definiția clasei DoubleArray este similară clasei IntArray.

Tot ca tipuri de date necesare pentru reprezentarea grafului se mai definesc clase pentru vectori de coordonate, cu două, trei, și patru dimensiuni: clasele Vect2, Vect3, Vect4, toate derivate dintr-o clasă de bază Vect, ale căror date și funcții membre mai importante sunt date mai jos:

```

class Vect : public Object{
    double *vect;           // vectorul de date
    int d;                  // dimensiune vector
public:
    Vect(int s, int *p);    // constructor
    virtual ~Vect(){delete vect;}
    double& operator[](int i){return vect[i];}
    Vect operator=(Vect v);
    Vect operator+=(Vect v);
    double *Get() const {return vect;}
    double Get(int i) const {return vect[i];}
    void Set(int i, double v){vect[i]=v;}
    void Set(double *v);
};

Vect::Vect(int s, double *p){
    d = s;
    vect = new double [d];
    for (int i=0; i<d; i++)
        vect[i] = p[i];
}

class Vect2 : public Vect {
public:
    Vect2(double *p); Vect(2,p){ }
    Vect2 operator=(Vect2 v);
};

```



```

    Vect2 operator+=(Vect2 v);
};
class Vect3 : public Vect {
public:
    Vect3(double *p) : Vect(3, p){ }
    Vect3 operator=(Vect3 v);
    Vect3 operator+=(Vect3 v);
};

```

Aceste clase definite în proiect pot fi înlocuite cu clase similare ale bibliotecii de clase a compilatorului folosit sau cu clase template. De exemplu, dacă se folosește compilatorul Microsoft Visual C++ (versiuni 4.2, 5.0, 6.0), atunci se pot prelua clase din biblioteca de clase MFC 3.0 (*Microsoft Foundation Class*), care implementează tipurile de colecții de date. Clasa *Object* corespunde (cu multe simplificări) clasei *CObject* din MFC, clasa *ObArray* corespunde clasei *CObArray*, clasa *IntArray* corespunde clasei *CUIIntArray*.

#### 11.2.1.2 Nodurile grafului

Nodurile grafului scenei sunt reprezentate printr-o ierarhie de clase de obiecte care modelează comportarea acestora și relațiile de moștenire dintre ele. Clasa de bază pentru toate nodurile grafului este clasa *Node*, care este o clasă abstractă, iar toate celelalte clase care descriu nodurile grafului sunt derivate din aceasta.

```

class Node : public Object{
    ObArray parents;
    Vect3 bmin; // volumul de delimitare
    Vect3 bmax; // (bounding box)
public:
    Node(){}
    ~Node();
    int Invisible();
    virtual void Draw()=0;
};

```

Clasa *Node* descrie comportarea generală a unui nod în graf prin definirea unui vector de pointeri la nodurile părinte (*parents*) și o funcție virtuală pură de redare (funcția *Draw()*), care se va redefini în fiecare clasă concretă derivată din clasa *Node*. Numărul de părinți ai unui nod este folosit în operațiile de ștergere a nodurilor grafului: un nod eliminat dintr-o poziție a grafului nu poate fi șters din memorie decât dacă numărul de părinți este egal cu zero.

Tot în clasa *Node* este implementat mecanismul de eliminare a obiectelor la redare prin testarea volumului de delimitare. Volumul de delimitare este definit în acest proiect ca un paralelipiped dreptunghic dat prin coordonatele minime și maxime pe cele trei axe de coordonate (grupate în vectorii *bmin* și *bmax*).

În cursul redării imaginii, dacă volumul de delimitare este sigur invizibil (se află în afara volumului de vizualizare), atunci nodul respectiv și toți fiii lui sunt



invizibili și sunt părăsiți. Funcția `Invisible()` a clasei `Node` returnează 1, dacă volumul de delimitare al nodului este complet în afara volumului de vizualizare, și 0 în celelalte cazuri.

Din clasa `Node` sunt derivate mai multe tipuri de clase, dintre care cele mai importante sunt: clasa `Material`, care definește aspectul obiectelor, clasa `Shape`, care definește forma obiectelor, clasa `Group`, care grupează mai multe obiecte, etc.

**Reprezentarea aspectului obiectelor.** Pentru descrierea aspectului obiectelor se definește materialul și caracteristicile de texturare. În proiectul CSV s-a introdus doar proprietatea de material. Proiectul poate fi extins cu ușurință și pentru texturare.

Materialul este descris de clasa `Material`, care definește coeficienții de reflectanță ambientală, difuză și speculară ai unui material dat printr-un nume unic în program.

```
class Material : public Object{
    char* name; // nume material
    GLfloat diffuse[4]; // componenta de difuzie
    GLfloat ambient[4]; // componenta ambientală
    GLfloat specular[4]; // componenta speculară
    GLfloat shininess; // exponent specular
public:
    Material(char* n = ""){ // material implicit
        name = new char(strlen(n)+1);
        strcpy(name, n);
        for (int i=0; i<4; i++){
            diffuse[i] = 1.0;
            ambient[i] = 0.0;
            specular[i] = 1.0;
            shininess = 20;
        }
    }
    Material(Material &m);
    ~Material(){delete name;}
    char* GetName(){return name;}
    float* GetAmbient(){return ambient;}
    float* GetDiffuse(){return diffuse;}
    float* GetSpecular(){return specular;}
    float GetShininess(){return shininess;}
};
```

Un obiect din clasa `Material` (de fapt, un pointer la un astfel de obiect), este folosit ca atribut al unei mulțimi de fețe ale unui obiect (`IndexedFaceSet`). La întâlnirea unui obiect din clasa `Material` în cursul traversării bazei de date, materialul respectiv este setat ca material curent al bibliotecii de redare (în mod obișnuit `OpenGL`).



**Reprezentarea formei obiectelor.** Modul cel mai obișnuit de reprezentare a obiectelor tridimensionale în scenele virtuale este reprezentarea poligonală. În exemplul implementat, un astfel de obiect tridimensional este descris de clasa Shape, derivată din clasa Node și conține listele de vârfuri, normale, coordonate de texturare, fețe și încă mai multe alte informații.

```
class Shape : public Node{
    ObArray coords;
    ObArray normals;
    ObArray texCoords;
    ObArray colors;
    ObArray faceSet;
    Vect3 center;
public:
    Shape(){}
    Shape(double s, int numcoord, int numfaces,
           int numvert, int *f, double *c,
           double* n=0, double* t=0, double* col=0);

    Shape(char *fn);
    ObArray* GetCoords() {return &coords;}
    ObArray* GetNormals() {return &normals;}
    ObArray* GetTexture() {return &texCoords;}
    ObArray* GetColors() {return &colors;}
    IndexedFaceSet* GetFaceSet() {
        int index = faceSet.GetSize()-1;
        return (IndexedFaceSet*)faceSet.GetAt(index);
    }
    void AddCoord(Vect3* pVect3){
        coords.Add(pVect3);
        center+=*pVect3;
    }
    void AddFaceSet(IndexedFaceSet* pSet){
        faceSet.Add(pSet);
    }
    void ReadObj(char *fileName);
    void Draw();
};
```

Clasa Shape descrie o rețea de poligoane care poate reprezenta o suprafață de frontieră închisă sau deschisă a unui solid. Impunerea condițiilor de închidere sau orientare consistentă a suprafeței reprezentate prin rețeaua de poligoane depinde de scopul aplicației și poate fi o funcție de verificare inclusă în constructorul clasei Shape.

Vectorii de coordonate ale vârfurilor (coords) și de normale (normals) sunt compuși din pointeri la obiecte de tip Vect3. Vectorul de coordonate de texturare (texCoords) este un vector de pointeri la clasa Vect2, iar vectorul de culori (colors) este un vector de pointeri la clasa Vect4. Fețele unui obiect



Shape sunt reprezentate printr-un vector (faceSet) de pointeri la clasa IndexedFaceSet. Clasa IndexedFaceSet descrie o mulțime de fețe ale unui obiect, grupate datorită unei proprietăți comune (de exemplu, material comun).

```
class IndexedFaceSet:public Object{
    int mode;                // mod de redare
    Material *material;      // materialul
    ObArray faces;           // lista de fețe
public:
    IndexedFaceSet(
        material = NULL;
    )
    IndexedFaceSet(char *matName);
    void AddFace(IndexedFace* pFace){
        faces.Add(pFace);
    }
    void Draw(ObArray *coords, ObArray *normals,
              ObArray *texCoords, ObArray *colors);
    .....
};
```

O față din lista de fețe (faces) este descrisă de clasa IndexedFace, care conține listele de indici reprezentate ca vectori de numere întregi (din clasa IntArray).

```
class IndexedFace:public Object{
    IntArray coordIndex;      // indici de coordonate
    IntArray normalIndex;     // indici de normale
    IntArray texCoordIndex;   // indici de coord. textura
    IntArray colorIndex;      // indici de culoare
public:
    IndexedFace(){ mode = MODE_NONE; }
    void AddCoordIndex(int index){
        coordIndex.Add(index);
    }
    void SetMode(int m){ mode |= m; }
    void ResetMode(int m){ mode &= ~m; }
    int GetMode(){ return mode; }
    void Draw(ObArray *coords, ObArray *normals,
              ObArray *texCoords, ObArray *colors);
    .....
};
```

O formă tridimensională (obiect din clasa Shape) se construiește folosind unul din constructorii clasei Shape. Construcția se realizează prin definirea conținutului vectorilor de date (coordonate, normale, fețe, etc) prin citirea unor fișiere de un anumit tip (ReadObj(), pentru format *obj* al bazei de date, ReadVr1(), pentru fișiere VRML, etc.).



În fig. 11.2 este prezentată ierarhia claselor care descriu un obiect poligonal. Unele clase sunt descrise simplificat. De exemplu, vectorii `coords`, `normals`, `texCoords`, `colors` nu conțin direct coordonatele ci pointeri la obiecte de tip `Vect`, care conțin coordonatele respective. De asemenea, vectorii de indici pot conține mai mult de trei indici în cazul general.

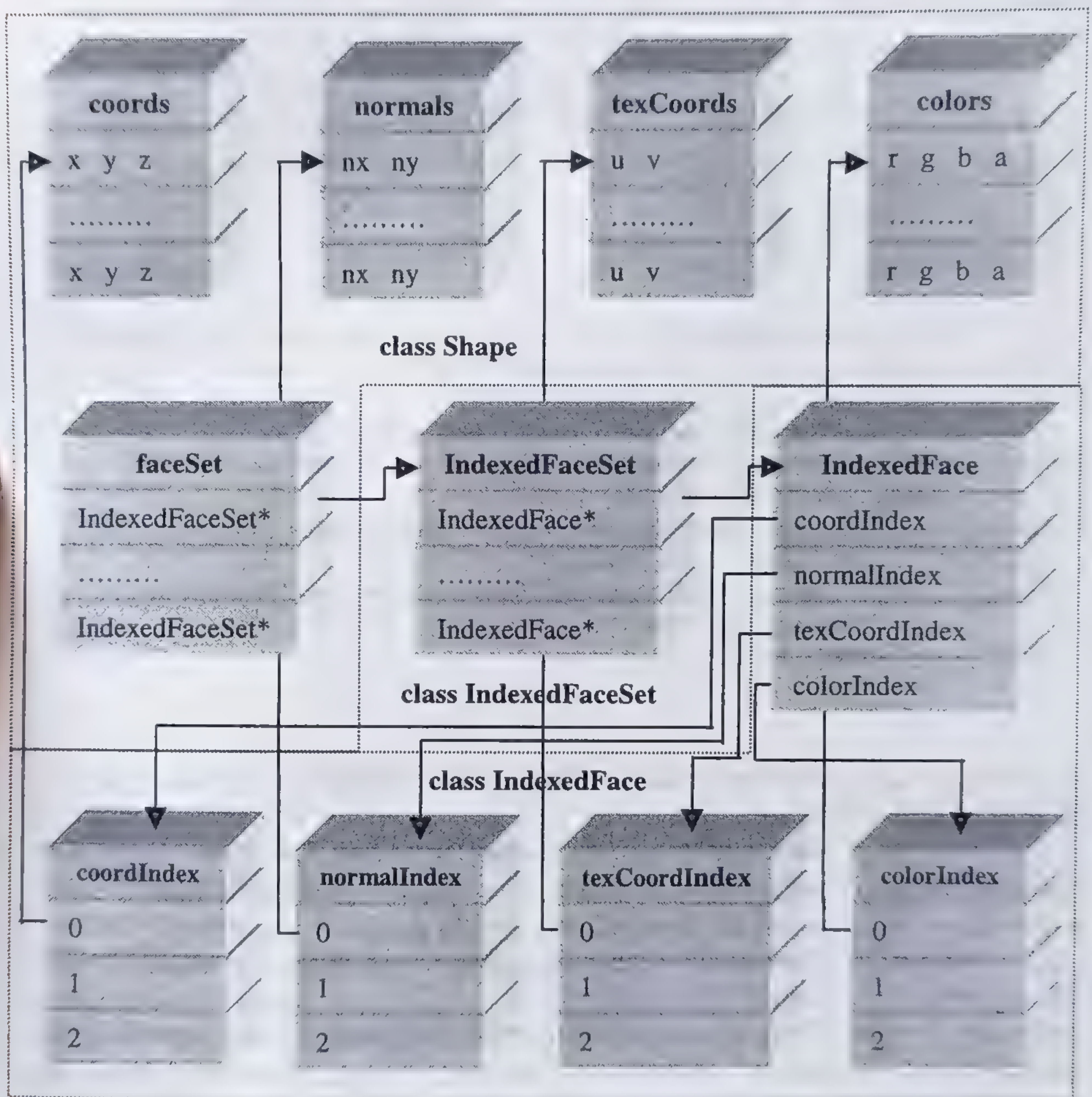


Fig. 11.2 Clasele de reprezentare a unui obiect poligonal:  
Shape, IndexedFaceSet și IndexedFace.

Din clasa `Shape` sunt derivate clase care descriu forme geometrice particulare putând fi generate direct sau prin calcul. De exemplu, clasa `Cube` modelează un cub printr-o mulțime de fețe indexate.



```

class Cube : public Shape{
    double size;
public:
    Cube(double s = 1.0);
};

double CubeCoords[8][3]={
    -1,-1, 1, 1,-1, 1,
    1,-1,-1,-1,-1,-1,
    -1, 1, 1, 1, 1, 1,
    1, 1,-1,-1, 1,-1, };

int CubeIndexedFace[6][4]={
    3,2,1,0,
    4,5,6,7,
    0,1,5,4,
    2,3,7,6,
    1,2,6,5,
    0,4,7,3 };

Cube::Cube(double s){ // constructor de inițializare
    size = s;
    for (int i=0;i<8;i++){
        AddCoord(new Vect3(s,CubeCoords[i]));
    }
    IndexedFaceSet* pSet = new IndexedFaceSet();
    for (i=0;i<6;i++){
        IndexedFace* pFace = new IndexedFace();
        for (int j=0;j<4;j++){
            pFace->AddCoordIndex(CubeIndexedFace[i][j]);
        }
        AddFaceSet(pSet);
        pSet->AddFace(pFace);
    }
}

```

În mod asemănător sunt definite clase pentru modelarea sferelor, cilindrilor, piramidelor, etc.

Ca exemplu de modul în care se creează un nod de descriere a unui obiect poligonal în graful scenei virtuale (clasa Shape) se consideră formatul obj al unei baze de date, care este un format foarte simplu și ușor de interpretat.

Formatul obj reprezintă fețele poligoanelor prin indecși în lista de vârfuri ale obiectului. Acesta este cel mai utilizat mod de reprezentare a modelului poligonal al obiectelor și este regăsit în majoritatea formatelor sau limbajelor de modelare. În continuare este prezentat modelul poligonal al unui cub cu latura 2, cu centrul în centrul sistemului de referință de modelare, în format obj.

```

#-- Modelul unui cub în format obj-- Fișier cub.obj
#-- Vectorul de vârfuri
#-- 'v' definește un vârf
v -1.00000 -1.00000 1.00000
v 1.00000 -1.00000 1.00000
v 1.00000 -1.00000 -1.00000
v -1.00000 -1.00000 -1.00000

```



```

v    -1.00000  1.00000  1.00000
v     1.00000  1.00000  1.00000
v     1.00000  1.00000 -1.00000
v    -1.00000  1.00000 -1.00000
#--
#--  Vectorul de normale
#--  'vn' definește normala într-un vârf
vn   -0.57803 -0.57803  0.57803
vn    0.57803 -0.57803  0.57803
vn    0.57803 -0.57803 -0.57803
vn   -0.57803 -0.57803 -0.57803
vn   -0.57803  0.57803  0.57803
vn    0.57803  0.57803  0.57803
vn    0.57803  0.57803 -0.57803
vn   -0.57803  0.57803 -0.57803
#--
#--  Vectorul de fețe
#--  'f' definește o față dată prin vârfuri
#--  fiecare vârf este definit printr-un grup de 1,2,
#--  sau 3 indecsi separați prin slash pentru
#--  coordonate, coordonate de textură și normală
#--  Coordonata de textură și (sau) normala poate lipsi
#--  Indicii în vector încep de la 1 (nu de la 0)
#--  Fețele următoare au date coordonatele și normalele
f    4//4  3//3  2//2  1//1
f    5//5  6//6  7//7  8//8
f    1//1  2//2  6//6  5//5
f    3//3  4//4  8//8  7//7
f    2//2  3//3  7//7  6//6
f    1//1  5//5  8//8  4//4

```

În formatul obj se definesc trei vectori de coordonate ai unui obiect: vectorul coordonatelor vârfurilor, vectorul normalelor și vectorul coordonatelor de texturare. Fiecare față poligonală este o listă de vârfuri definite prin indecși în vectorul coordonatelor vârfurilor, în vectorul coordonatelor de textură și în vectorul normalelor.

Prin citirea unui fișier obj se creează un nod de tipul Shape care reprezintă descrierea poligonală a obiectului. De exemplu, se pot crea noduri cu diferite obiecte prin citirea fișierelor corespunzătoare:

```

Shape *pObject1 = new Shape("../cub.obj"); // cub
Shape *pObject1 = new Shape("../f-16.obj"); // avion F-16

```

**Reprezentarea nodurilor de grupare.** Nodurile de grupare sunt nodurile care admit unul sau mai multe noduri fii în graful scenei. Clasa de bază pentru toate nodurile de grupare este clasa Group, derivată din clasa Node și din ea sunt derivate clasele Transform, LOD, Switch, Billboard.



*Clasa Group* definește un vector de pointeri la alte noduri, care devin astfel noduri fii ale nodului respectiv:

```
class Group : public Node{
    ObArray childs;
public:
    void AddChild(Node* pNode){
        childs.Add(pNode);
    }
    Node* GetChild(int i){
        return (Node*)childs.GetAt(i);
    }
    int GetSize(){return childs.GetSize();}
    void Draw();
};
```

*Clasa Transform* este derivată din *clasa Group* și permite introducerea unei transformări printr-o matrice de transformare care se aplică tuturor fiilor acestui nod:

```
class Transform : public Group{
    double matrix[16];
public:
    Transform(); // constructor implicit
    Transform(double *m); // inițializare
    void Scale(double x, double y, double z);
    void Rotate(double angl, double x, double y, double z);
    void Translate(double x, double y, double z);
    void Draw();
};
```

Matricea de transformare *matrix* memorează o matrice  $4 \times 4$  în modul coloană majoră. Ea este inițializată cu matricea identitate de către constructorul implicit al clasei și poate fi modificată prin funcțiile membre de translație, rotație, scalare.

*Clasa LOD (level of detail)* introduce un obiect reprezentat cu nivele de detaliu multiple. Modelarea cu nivele de detaliu multiple și selecția unui nivel la redare reprezintă una din cele mai importante metode de menținere a unei viteze constante de generare a imaginii scenelor virtuale complexe. Orice sistem grafic are o capacitate finită de redare a primitivelor grafice și menținere a unei frecvențe constante a cadrelor imaginii (număr de cadre/secundă) necesită menținerea unui număr cât mai redus de poligoane de reprezentare a obiectelor, asigurând în același timp realismul vizual dorit.

Posibilitatea prelucrării cu nivele de detaliu multiple a obiectelor este dată de caracteristica proiecției perspective, conform căreia dimensiunea proiecției unui obiect scade odată cu creșterea distanței acestuia față de punctul de observare ( $z_v$ ). Acest lucru se poate observa din relația (4.10) care este reprodusă în continuare:



$$x_N = \frac{d x_v}{g z_v}, \quad y_N = \frac{d y_v}{h z_v}$$

Detaliile obiectelor aflate la distanță mare față de punctul de observare nu mai sunt vizibile (imaginea lor putând ajunge sub dimensiunea unui pixel), de aceea aceste obiecte se pot reprezenta cu un număr mai mic de poligoane. Prelucrarea cu nivele de detaliu multiple constă în modelarea unui obiect printr-o succesiune de reprezentări din ce în ce mai simple (cu precizie și număr de poligoane scăzut) și selectarea versiunii de reprezentare în funcție de distanță.

Nivelele de detaliu ale unui obiect se organizează ca un grup de reprezentări, și, pentru fiecare dintre ele se specifică intervalul de distanțe între care se va selecta la redare. În cursul redării imaginii se calculează distanța de la punctul de observare la centrul obiectului și, în funcție de aceasta, se selectează reprezentarea adecvată.

O astfel de prelucrare este implementată printr-un nod în graful scenei de tipul LOD, care este derivat din clasa Group, deci admite un număr oarecare de noduri fii, fiecare nod fiu fiind o reprezentare pe un anumit nivel de detaliu. Clasa LOD conține un vector de distanțe (range), de dimensiune egală cu numărul de nivele de detaliu (nodurile fii). Fiecare valoare din vectorul de distanțe reprezintă un punct de comutare între nivele. Nivelul de detaliu 0 (cu precizia maximă) este vizibil de la distanța 0 până la distanța memorată în primul element al vectorului de distanțe (range[0]); celelalte nivele sunt vizibile între două valori consecutive memorate în vectorul de distanțe; ultimul nivel este vizibil până la o distanță maximă (ultima valoare din vectorul range); peste această distanță obiectul nu mai este vizibil.

```
class LOD : public Group {
    Vect3 center;
    DoubleArray range;
public:
    LOD() : center(0.0, 0.0, 0.0) {}
    LOD(int n, double* r);
    double Dist();
    void Draw();
};
LOD::LOD(int n, double* r) : center(0.0, 0.0, 0.0) {} {
    for (int i=0; i<n; i++) {
        range.Add(r[i]);
    }
}
```

La sfârșitul acestui subcapitol sunt date mai multe exemple de construire și redare a scenelor virtuale, inclusiv prelucrarea nivelelor de detaliu multiple.

În fig. 11.3 este prezentată ierarhia principalelor clase de reprezentare a nodurilor grafului scenei virtuale în proiectul CSV. Clasa Light introduce o sursă de lumină în graful scenei.



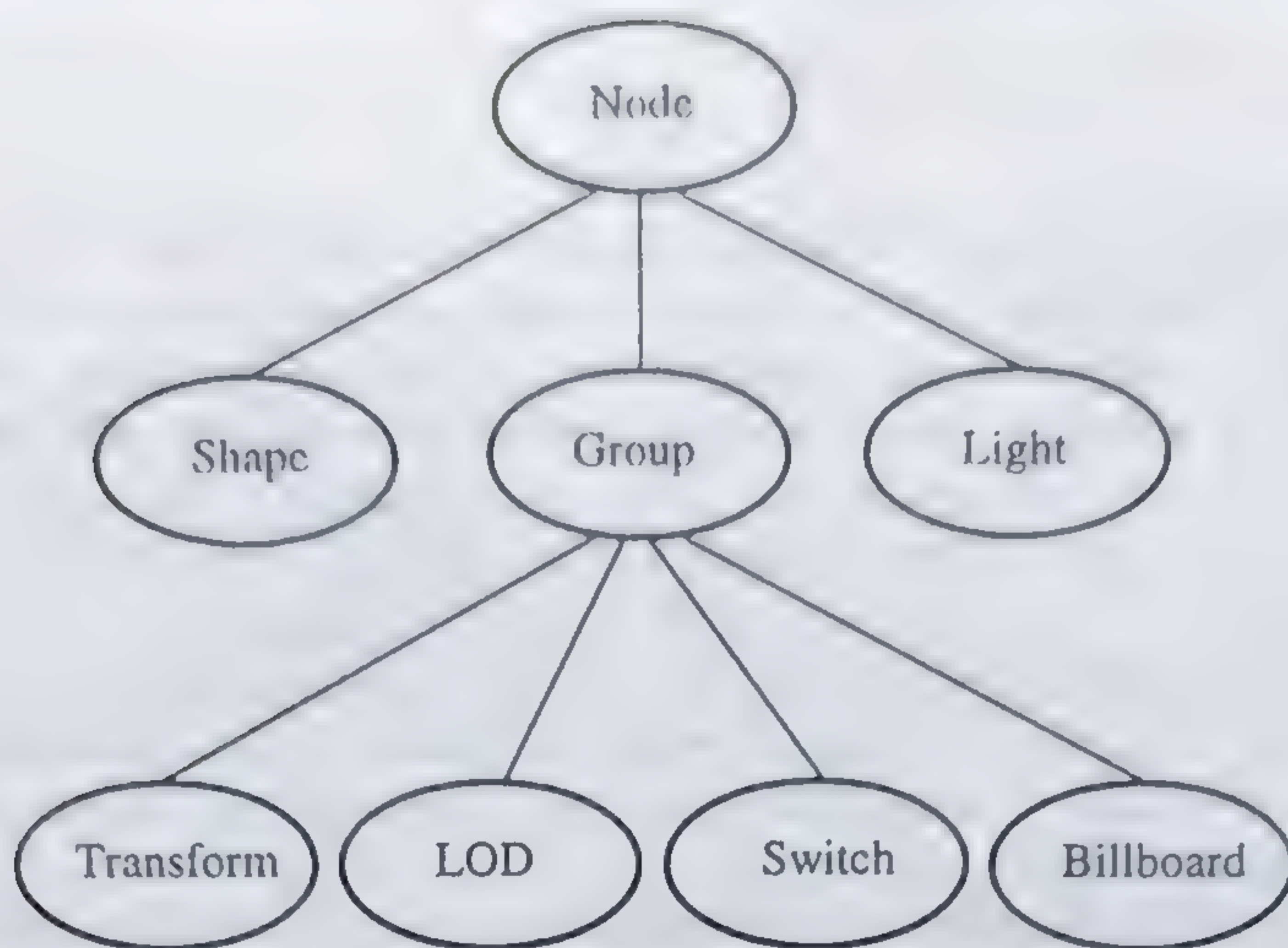


Fig. 11.3 Ierarhia claselor de descriere a grafului scenei.

### 11.2.2 REDAREA SCENELOR VIRTUALE

Graful de reprezentare a unei scene virtuale se creează prin citirea unui sau mai multor fișiere de date. În exemplele din text s-au folosit fișiere în format obj, dar modelul în format intern al scenei (graful scenei) este același, indiferent de formatul bazei de date încărcate.

În cursul operației de parcurgere în adâncime (traversare) a grafului scenei, în fiecare nod vizitat se apelează funcția de redare a nodului, care în proiectul CSV este numită funcția `Draw()`. Această funcție testează mai întâi vizibilitatea volumului de delimitare și, dacă acesta este invizibil funcția se termină și nodul împreună cu fii lui sunt ignorați (operație care se numește “rețezarea” nodului). Dacă nodul nu este rețezat, se apelează recursiv funcțiile de redare a nodurilor descendente și, pentru unele din tipurile de noduri de grupare, se actualizează matricea curentă a stivei matricelor de transformări, pentru instanțierea (plasarea) obiectelor în sistemul de coordonate universal.

Un obiect tridimensional reprezentat poligonal prin clasa `Shape` este întotdeauna nod frunză al grafului, deoarece nu poate conține noduri fii. Pentru redarea imaginii unui astfel de obiect se apelează funcția `Draw()`, membră a clasei `Shape`. Funcția `Draw()` a clasei `Shape` apelează funcția membră `Draw()` a clasei `IndexedFaceSet` pentru fiecare mulțime de fețe indexate. Aceasta, la rândul ei, apelează funcția membră `Draw()` a clasei `IndexedFace` pentru fiecare față. Funcția `Draw()` a unei fețe (clasa `IndexedFace`) conține toate apelurile de funcții OpenGL, ca `glVertex#()`, `glNormal#()`, `glTexCoordinate#()`, depinzând de modul de redare a obiectului. Funcțiile `Draw()` ale claselor de reprezentare a unei forme poligonale arată astfel:



```

void Shape::Draw(){
    if(Invisible()) // testare volum de delimitare
        return; // retezare (culling)
    int size = faceSet.GetSize();
    for (int i=0;i<size;i++){
        IndexedFaceSet* pSet =
            (IndexedFaceSet*)faceSet.GetAt(i);
        pSet->Draw(&coords, &normals,
            &texCoords, &colors);
    }
}

void IndexedFaceSet::Draw()(ObArray *coords,
    ObArray *normals, ObArray *texCoords,
    ObArray *colors){
    int size = faces.GetSize();
    if (material){
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
            material->GetAmbient());
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,
            material->GetDiffuse());
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
            material->GetSpecular());
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS,
            material->GetShininess());
    }
    for (int i=0;i<size;i++){
        IndexedFace* pFace= (IndexedFace*)faces.GetAt(i);
        pFace->Draw(coords,normals,texCoords,colors);
    }
}

void IndexedFace::Draw(ObArray *coords,
    ObArray*normals,ObArray*texCoords,ObArray *colors) {
    int size = coordIndex.GetSize();
    glBegin(GL_POLYGON);
    for (int i=0;i<size;i++){
        int index = coordIndex.GetAt(i);
        Vect3* coord = (Vect3*)coords->GetAt(index);
        if (mode & MODE_SMOOTH){
            index = normalIndex.GetAt(i);
            Vect3* norm = (Vect3*)normals->GetAt(index);
            glNormal3dv(norm->Get());
        }
        if (mode & MODE_TEXTURE){
            index = texCoordIndex.GetAt(i);
            Vect2* tex = (Vect2*)texCoords->GetAt(index);
            glTexCoord2dv(tex->Get());
        }
        glVertex3dv(coord->Get());
    }
    glEnd();
}

```



La traversarea unui nod de tip Group trebuie să se asigure condiții de instanțiere identice fiecărui nod fiu al grupului. De aceea, matricea de transformare curentă (din capul stivei MODELVIEW) trebuie să fie salvată înaintea traversării fiecărui fiu și restaurată după traversarea acestuia. Funcția membră Draw() a clasei Group este următoarea:

```
void Group::Draw(){
    if(Invisible())    // testare volum de delimitare
        return;      // retezare (culling)
    int size=childs.GetSize();
    for (int i=0;i<size;i++){
        glPushMatrix();
        GetChild(i)->Draw();
        glPopMatrix();
    }
}
```

Clasa Transform redă mulțimea de noduri fii după aplicarea transformării definite de matricea nodului, prin înmulțirea acesteia cu matricea curentă din stiva matricelor de modelare-vizualizare. Funcția Draw() a clasei Transform este:

```
void Transform::Draw(){
    glMultMatrix(matrix);
    Group::Draw();
}
```

La traversarea unui nod din clasa LOD se selectează nodul fiu corespunzător distanței centrului obiectului față de punctul de observare. Funcția Draw() a clasei LOD realizează acest lucru:

```
void LOD::Draw(){
    double d = Dist();    // calculează distanța
                        // fata de punctul de observare
    int size = range.GetSize();
    for (int i=0;i<size;i++){
        if (d <=range[i])
            break;
    }
    if (i <size){
        printf("LOD:%d  Dist:%f\n",i,d);
        glPushMatrix();
        GetChild(i)->Draw();
        glPopMatrix();
    }
}
```

Toate aceste funcții de redare (funcțiile Draw()) ale claselor nodurilor din graful scenei implementează algoritmul de traversare a grafului scenei prin parcurgerea în adâncime a grafului combinată cu actualizarea stării matricei



curente de transformare, prezentat principal la începutul acestui subcapitol. Selecția operațiilor executate în funcție de tipul nodului se face prin faptul că funcțiile de redare sunt funcții membre ale diferitelor clase de reprezentare a nodurilor.

În continuare sunt prezentate câteva exemple de construire și redare a obiectelor și scenelor virtuale ierarhice.

### ■ Exemplul 11.1

În acest exemplu se creează modelul poligonal al diferitelor obiecte prin citirea unor fișiere de date în format obj. Funcția `Display()` a programului apelează funcția de redare `Draw()` a nodului rădăcină al grafului scenei (`pNode`).

```
#include "../Object.h"
static Node* pNode; //nodul rădăcină al grafului scenei

Node* BuildScene1() {
    Shape *pShape;
    pShape = new Shape("../rose.obj"); // model vaza
    //pShape = new Shape("../porsche.obj"); // model masina
    //pShape = new Shape("../flowers.obj"); // model flori
    return pShape;
}

void Init(){
    pNode = BuildScene1();
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
}

void Display(){
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotated(-45, 0, 1, 0);
    glTranslated(-50, -10, -50);
    glScaled(0.3, 0.3, 0.3);
    pNode->Draw();
    glPopMatrix();
    glutSwapBuffers();
}
```

Funcțiile `Reshape()` și `main()` sunt aceleași ca în programele precedente. În funcția `Init()` este apelată funcția de construcție a scenei virtuale `BuildScene1()`. În acest exemplu scena constă dintr-un singur nod de tipul `Shape`, care reprezintă un obiect poligonal a cărui descriere se încarcă dintr-un



fișier de date în format obj. La execuția acestui program se afișează imaginea obiectului încărcat: o vază cu un trandafir, o mașină Porsche sau un ghiveci cu flori (fig. 11.4).



Fig. 11.4 Redarea obiectelor poligonale: (a) rose.obj (2246 poligoane); (b) porsche.obj (4740 poligoane); (c) flowers.obj (4061 poligoane).

Scene virtuale complexe, compuse din numeroase obiecte, se pot construi prin conectarea în graf a diferitelor tipuri de noduri. În general, structura grafului se creează prin citirea fișierelor bazei de date reprezentată într-un format care descrie organizarea ierarhică a scenei, cum sunt formatele flt, dwb sau limbajul VRML. În proiectul simplu dezvoltat în acest capitol, graful scenei virtuale poate fi creat prin program, ca în exemplele următoare.

### ■ Exemplul 11.2

În acest exemplu se construiește o scenă compusă din patru obiecte avioane F-16, plasate în diferite poziții în scenă. În funcția `Init()` se apelează o altă funcție de construcție a scenei, `BuildScene2()`, care creează un graf al scenei compus din patru obiecte F-16 amplasate în diferite poziții în scenă astfel:

```
Node* BuildScene2(){
    Group *pG = new Group;
    Shape *pO = new Shape("../f-16.obj");
    Transform* pT;
    // Prima instantiere
    pT = new Transform;
    pT->AddChild(pO);
    pG->AddChild(pT);
    // A doua instantiere
    pT = new Transform;
    pT->Translate(0.0, 8.0, -40.0);
    pT->Rotate(20.0, -90.0, -10.0);
```



```

    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    // A treia instantiere
    pT = new Transform;
    pT->Translate(30.0,16.0,-40.0);
    pT->Rotate(20.0,90.0,45.0);
    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    // A patra instantiere
    pT = new Transform;
    pT->Translate(-30.0,16.0,-40.0);
    pT->Rotate(20.0,-90.0,-45.0);
    pT->Scale(2.0,2.0,2.0);
    pT->AddChild(pO);
    pG->AddChild(pT);
    return pG;
}

```

Graful scenei create în funcția `BuildScene2()` este prezentat în fig. 11.5.

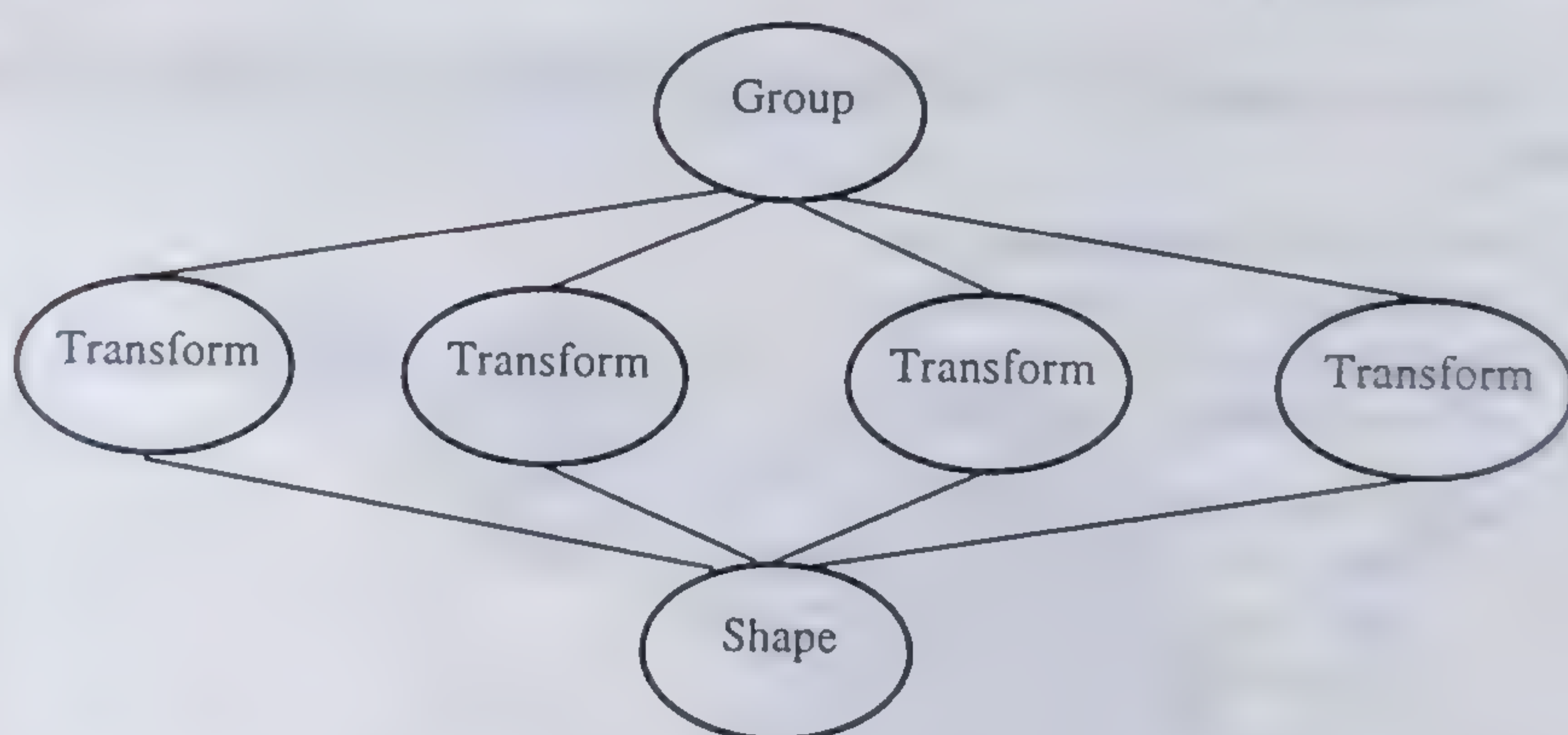


Fig. 11.5 Graful scenei virtuale construit de funcția `BuildScene2()`.

Nodul rădăcină al grafului este un nod de tip `Group`, care are 4 noduri fii. Fiecare nod fiu este un nod de tip `Transform`, care definește poziția sistemului de referință model în sistemul de referință universal. Fiecare nod de transformare are câte un singur nod descendent, și anume nodul de tip `Shape` care modelează avionul F-16.

Imaginea afișată la execuția programului este dată în fig. 11.6. Axele de coordonate ale sistemului universal sunt reprezentate prin apelul funcției `Axes()` (prezentată într-unul din programele precedente) în funcția `Display()`.





Fig. 11.6 Imaginea scenei virtuale create prin instanțierea a patru obiecte F-16.

### Exemplul 11.3

În acest exemplu se prezintă crearea și prelucrarea nivelelor de detaliu multiple.

```
#include <iostream.h>
#include "../Objects.h"
double xv = 0, yv = 0, zv = 0;
static Node* pNode;

Node* BuildScene3(){
    double r[3]={10.0,-16.0,32.0};
    LOD* pL = new LOD(3,r);
    pL->AddChild(new Sphere(1.25,32,32));
    pL->AddChild(new Sphere(1.25,24,24));
    pL->AddChild(new Sphere(1.25,14,14));

    Group *pG = new Group;
    Transform* pT = new Transform;
    pT->Translate(-4,-2,-26);
    pT->AddChild(pL);
    pG->AddChild(pT);
    pT = new Transform;
    pT->Translate(0,-2,-15);
    pT->AddChild(pL);
    pG->AddChild(pT);
    pT = new Transform;
    pT->Translate(2,-2,-8);
    pT->AddChild(pL);
```



```
pG->AddChild(pT);  
return pG;  
}  
void Init(){  
    pNode = BuildScene3();  
    glColor3f(0.2,0.2,0.2);  
}  
void Display(){  
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
    pNode->Draw();  
    glutSwapBuffers();  
}
```

În funcția `BuildScene3()` se creează graful scenei compus din trei instanțe ale unui obiect cu trei nivele de detaliu. Obiectul este o sferă și pentru fiecare nivel de detaliu se reprezintă printr-un număr diferit de fețe: nivelul 0 este reprezentat prin  $32 \times 32 = 1024$  de fețe, nivelul 1 prin  $24 \times 24 = 576$  fețe, nivelul 2 prin  $14 \times 14 = 196$  fețe. Distanțele de comutare între nivelele de detaliu sunt date în vectorul `range`. Selecția nivelului de detaliu se face în funcție de distanța centrului sferei față de punctul de observare. La execuția acestui program se obține imaginea din fig. 11.7 și se afișează la consolă mesajele:

```
LOD: 2   Dist: 26.381812  
LOD: 1   Dist: 15.152746  
LOD: 0   Dist: 8.485218
```

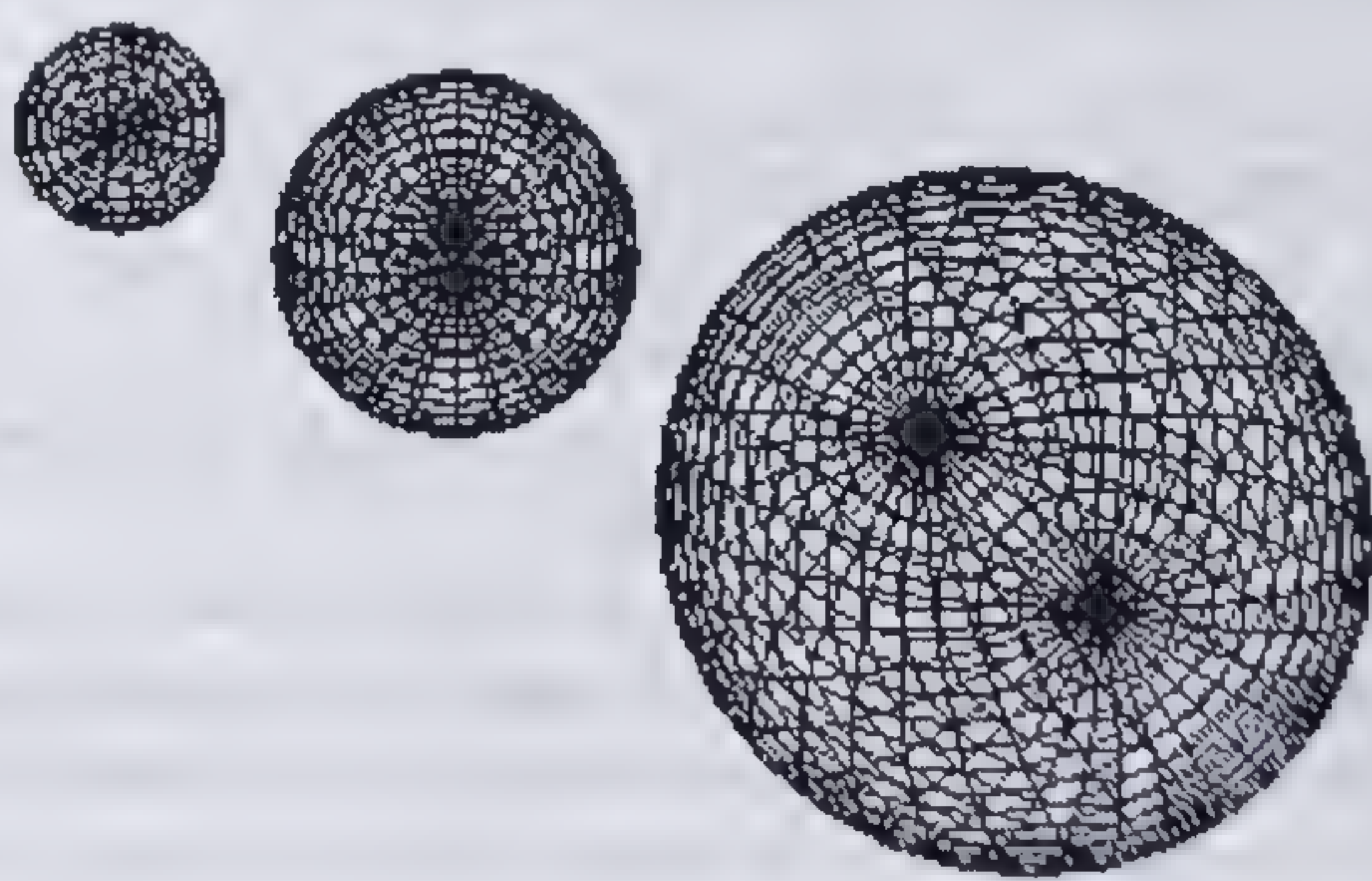


Fig. 11.7 Reprezentarea obiectelor cu nivele de detaliu multiple.

Din figura de mai sus reiese eficiența reprezentării obiectelor cu nivele de detaliu multiple. Sfera din stânga, aflată la distanța de 26.381, este redată pe nivelul 2 de detaliu, cu numai 196 de fețe. Sfera din dreapta, aflată la distanța de 8.485, este redată pe nivelul 0 de detaliu, cu 1024 de fețe. Și, cu toată diferența de precizie a aproximării poligonale, aspectul nivelului 2 de detaliu nu este inferior nivelului 0, datorită dimensiunii reduse a obiectului în proiecție perspectivă, atunci când distanța este mai mare. Reprezentarea obiectelor aflate la distanță mare cu precizie



ridicată produce un consum inutil de timp de calcul pentru generarea imaginii. Ca exercițiu, se poate încerca reprezentarea sferelor cu aceeași precizie indiferent de distanța față de punctul de observare. Se va observa că reprezentarea tuturor obiectelor cu precizie maximă nu aduce nici un beneficiu din punct de vedere al aspectului pentru obiectele depărtate, iar reprezentarea cu precizie minimă este inacceptabilă pentru obiectele apropiate. Reprezentarea cu nivele de detaliu multiple este o modalitate mult mai eficientă din punct de vedere al timpului de execuție și păstrează (în anumite limite) realismul imaginii. Graful scenei create de funcția `BuildScene3()` este dat în fig. 11.8.

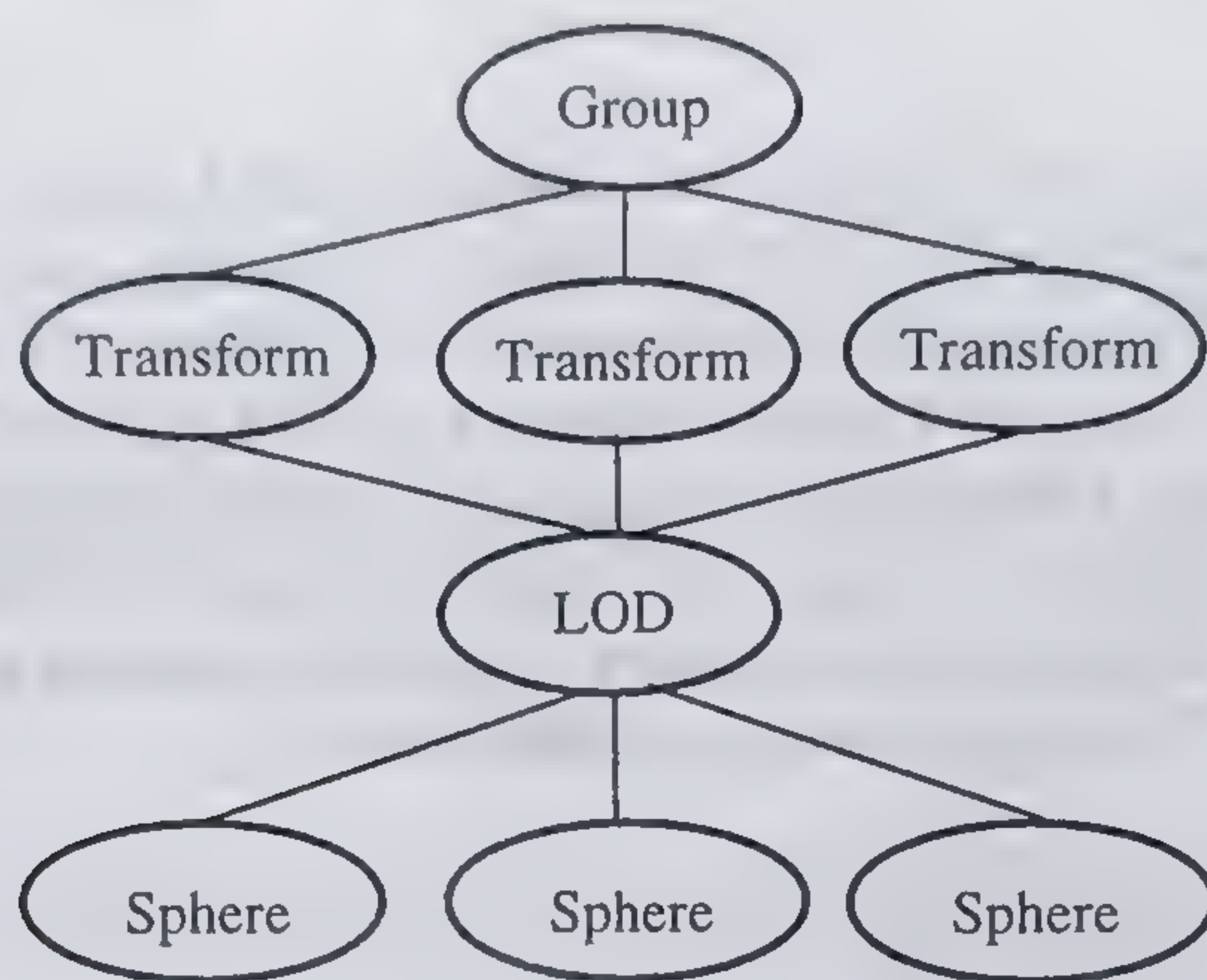


Fig. 11.8 Graful scenei create de funcția `BuildScene3()`.

În fig. 11.9 sunt prezentate două nivele de detaliu diferite ale aceluiași obiect, avionul F-16. În prima versiune sunt reprezentate 2413 poligoane; în cea de-a doua versiune sunt reprezentate 286 poligoane. Acesta este un model proiectat de firma Viewpoint, alături de alte numeroase modele de obiecte din cele mai variate domenii (elicoptere, mașini, clădiri, etc.).

Așa cum s-a mai menționat, abordarea la un nivel înalt a aplicațiilor grafice (crearea și redarea scenelor virtuale) diferă foarte mult de la o aplicație la alta, existând numeroase sisteme de dezvoltare (toolkit-uri), formate de baze de date sau limbaje de creare a scenelor virtuale. În acest subcapitol s-a încercat prezentarea unui proiect cât mai simplu de creare și redare a scenelor virtuale, dar care înglobează majoritatea caracteristicilor sistemelor complexe, cum sunt Performer, WorldUp, dVS, EasyEcene, Multigen.

Este de așteptat ca înțelegerea modalității principale de creare și redare a scenelor virtuale să ajute la abordarea și înțelegerea rapidă a oricărui astfel de sistem de dezvoltare. De asemenea, un astfel mod de organizare ierarhică, printr-un graf aciclic direcționat, a scenelor virtuale, corespunde formatului unei baze de date grafice descrise în limbajul VRML. Acest lucru se va observa cu ușurință în subcapitolul următor.



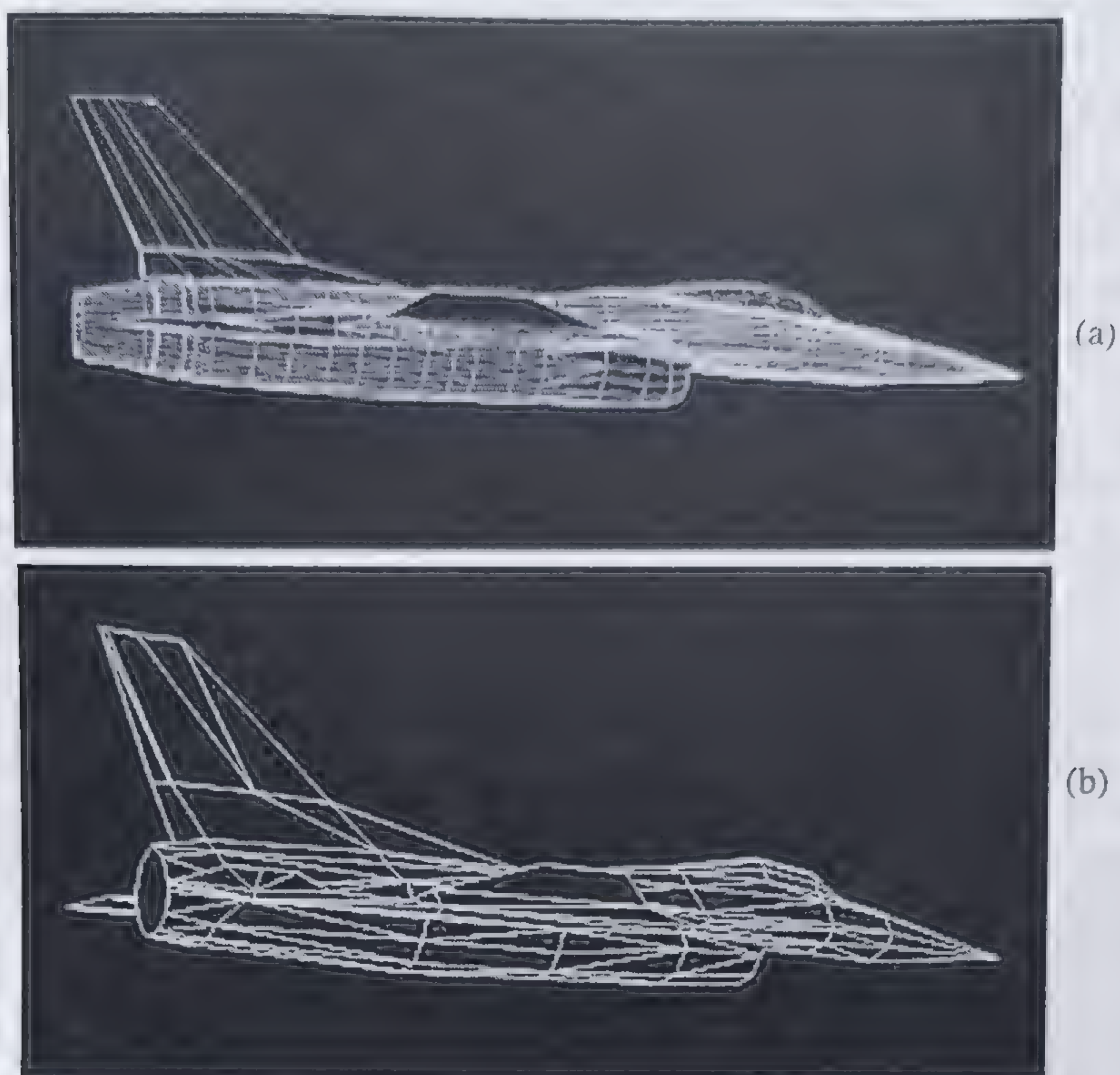


Fig. 11.9 Modelul avionului F-16 pe două nivele de detaliu diferite:  
(a) 2413 poligoane; (b) 286 poligoane.

### 11.3 LIMBAJUL VRML

Limbajul *VRML* (*Virtual Reality Modeling Language*) este un limbaj folosit în crearea scenelor virtuale tridimensionale, cu posibilitatea de acces a acestora prin rețeaua Internet WWW (*World Wide Web*). Versiunea folosită în momentul de față este VRML 2.0 (numită și VRML 97, fiind standard ISO din anul 1997) și este o revizuire a versiunii VRML 1.0, stabilite în 1995. Numeroase informații referitoare la limbajul VRML se pot găsi în rețeaua Internet (<http://www.vrml.org>).

În prima versiune, limbajul VRML 1.0 permitea crearea unei scene virtuale compuse din obiecte tridimensionale, dar cu posibilități limitate de interactivitate prin rețeaua Internet. VRML 1.0 a fost dezvoltat pe baza formatului de descriere a bazelor de date Open Inventor, elaborat de firma Silicon Graphics, care suportă descrierea scenelor ierarhice cu obiecte poligonale, lumini, materiale și efecte realiste. Un subset al formatului Open Inventor, cu extensii pentru acces prin



rețeaua Internet, a constituit baza de pornire pentru limbajul de descriere a scenelor virtuale VRML. În momentul de față, numeroase titluri de cărți, articole, reviste, documentație, exemple, adrese de grupuri de lucru, etc., se găsesc pe rețea despre limbajul VRML și utilizarea acestuia.

Un fișier VRML este un fișier ASCII care conține descrierea unei scenei virtuale și a unor acțiuni interactive, și poate fi accesat prin rețea folosind programe de navigare (browsere). Pentru a afișa o scenă VRML din rețeaua Internet sau dintr-un fișier de pe disc (cu extensia .vrl) este necesar un browser VRML, configurat ca o extensie (plug-in) al unui browser Web. Pentru browserul Netscape Navigator, există browserul Cosmo Player (de la compania Platinum Technology) pentru interpretarea fișierelor VRML. Un browser VRML citește fișierul care conține descrierea scenei virtuale, creează graful scenei și memorează datele de comportare a obiectelor și de interacțiuni corespunzătoare. Observarea scenei virtuale și navigarea în direcția dorită se realizează prin comenzi ale browserului. Aceste comenzi sunt foarte simple și se găsesc în documentația on-line a acestuia.

Deși utilizarea cea mai frecventă a limbajului VRML este în crearea și redarea scenelor 3D prin rețeaua Internet, un fișier VRML poate fi folosit și ca descriere a unei scene virtuale pentru aplicații de realitate virtuală care nu se desfășoară în rețea. În această situație, pentru citirea fișierelor VRML se folosește un program de interpretare a fișierelor VRML (*parser*) care creează graful scenei în concordanță cu structura programului de aplicație. În domeniul public pe rețeaua Internet sunt disponibile un mare număr de parsere care pot fi integrate într-o aplicație. De exemplu, există VRML 2.0 Parser de la Silicon Graphics (<http://vrml.sgi.com/>) sau Viper Parser (de la *NIST- National Institute of Standard and Technology*) implementat în limbajul Java.

În oricare situație, formatul VRML permite construirea grafului scenei, furnizând toate informațiile despre obiectele tridimensionale, ierarhia acestora, condiții de mediu ambiant, materiale, etc. Graful scenei, construit prin citirea unui sau mai multor fișiere VRML, este apoi traversat pentru obținerea imaginii obiectelor în diferite situații de navigare a scenei. Redarea imaginii obiectelor pe display se realizează prin intermediul bibliotecilor grafice de nivel mai scăzut, cum sunt OpenGL sau Direct3D (de la Microsoft). Aceste biblioteci asigură interfața corespunzătoare cu acceleratorul grafic al sistemului.

### 11.3.1 SPECIFICAȚIILE LIMBAJULUI VRML

Specificațiile limbajului VRML 97 descriu conceptele de realizare a scenelor virtuale, descrierea nodurilor, descrierea câmpurilor, a evenimentelor și interacțiunea între noduri. Similar limbajelor de programare standard, specificațiile VRML conțin reguli sintactice și reguli semantice ale limbajului.

Un fișier VRML conține un header (același pentru ISO VRML 97 și VRML 2.0), urmat de descrierea nodurilor grafului scenei, care specifică formele geometrice, grupurile, transformările și atributele mediului virtual ce se contruiește și vizualizează. Sintaxa VRML este simplă, cu cuvinte cheie în limba engleză sugestive pentru entitatea definită. Headerul fișierului este:



```
#VRML 2.0 utf8
# Alte comentarii
```

unde `utf8` este setul de caractere folosit. Este obligatorie introducerea versiunii VRML 2.0 imediat după caracterul `#` (fără nici un spațiu liber).

Un nod este specificat prin tipul nodului, urmat de un bloc cuprins între acolade. Blocul din interiorul acoladelor conține zero sau mai multe câmpuri (*fields*) și pentru fiecare câmp se specifică valoarea (sau valorile) acestuia (*field value*). Fiecare câmp al unui nod are un nume și admite una sau mai multe valori de un tip de date determinat de tipul câmpului.

În VRML este extinsă noțiunea de nod pentru orice entitate care grupează mai multe informații. Se poate observa diferența dintre noțiunea de *nod VRML* și noțiunea de nod într-un graf al scenei (cum au fost cele descrise în proiectul CSV). Un nod într-un graf este o componentă a unei structuri ierarhice, fiecare nod având unul sau mai mulți părinți (cu excepția nodului rădăcină care nu are nici un părinte) și unul sau mai mulți fii (cu excepția nodurilor frunză care nu au fii). Un nod VRML poate fi nod în graful scenei, sau poate să fie inclus (ca valoare a unui câmp) într-un alt nod. Modul cum se construiește graful scenei pornind de la specificarea nodurilor VRML depinde de programul de aplicație.

Sistemele de coordonate folosite în VRML sunt aceleași cu sistemele de coordonate din OpenGL (fig. 6.3): sistemul de referință universal este un sistem drept, sistemul de referință observator este de asemenea un sistem de referință drept, definit ca localizare și orientare relativ la sistemul de referință universal; proiecția perspectivă se definește printr-un plan perpendicular pe axa  $z$  a sistemului de referință de observare, cu direcția de observare spre  $-z$ .

Tipurile de noduri VRML sunt următoarele:

- **Forme și descrieri geometrice (*shapes*):** Box, Cone, Coordinate, Cylinder, EvaluationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, Normal, PointSet, Shape, Sphere, Text.
- **Aspect (*appearance*):** Appearance, Color, FontStyle, ImageTexture, Material, MovieTexture, PixelTexture, TextureCoordinate, TextureTransform.
- **Grupuri (*grouping*):** Anchor, Billboard, Collision, Group, Inline, LOD, Switch, Transform.
- **Mediu (*environment*):** AudioClip, Background, DirectionalLight, Fog, PointLight, Sound, SpotLight.
- **Vizualizare (*viewing*):** NavigationInfo, Viewpoint.
- **Animatie (*animation*):** ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator, TimeSensor.
- **Interacțiuni (*interaction*):** CylinderSensor, PlaneSensor, ProximitySensor, SphereSensor, VisibilitySensor, TouchSensor.
- **Alte noduri:** Script, WorldInfo.



În total sunt 54 de tipuri de noduri și fiecare tip admite unul sau mai multe tipuri de câmpuri. Tipurile de date prin care se specifică valoarea unui câmp pot fi: numere întregi, numere în virgulă flotantă, valori boolene, șiruri de caractere, sau pot fi folosite chiar noduri VRML. Un exemplu de fișier VRML foarte simplu este dat în continuare.

#### ■ Exemplul 11.4

```
#VRML 2.0 utf8
# Construiește un cilindru 3D
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0.0, 0.5, 1.0
    }
  }
  geometry Cylinder {
    height 2.0
    radius 1.5
  }
}
```

Nodul de tipul Shape introduce un obiect tridimensional pentru care se specifică câmpurile cu numele appearance și geometry, pentru definirea aspectului și a formei. Valoarea câmpului appearance este un nod de tipul Appearance, definit imediat prin blocul corespunzător între acolade. Nodul Appearance are specificat câmpul material a cărui valoare este dată printr-un nod de tipul Material. Nodul de tip Material conține câmpul diffuseColor, pentru care se specifică valoarea prin trei numere în virgulă flotantă, pentru componentele roșu, verde, albastru. Valoarea câmpului geometry este un nod de tipul Cylinder, care este definit prin câmpurile height și radius, cu valori numere în virgulă flotantă.

Un fișier VRML poate conține mii de noduri și fiecare nod conține un număr variabil de câmpuri, tipic două-trei câmpuri, dar pot fi și zeci de câmpuri. Din fericire, toate câmpurile nodurilor au prevăzute valori implicite rezonabile, astfel că nodurile pot fi folosite chiar dacă nu se specifică valoarea tuturor câmpurilor. De exemplu, valoarea implicită a culorii de ștergere este negru și, dacă nu se specifică altă culoare de fundal (printr-un nod Background), atunci la navigare prin intermediul browserelor fundalul rămâne negru. Poziția inițială a punctului de observare este la (0.0, 0.0, 10.0) și, dacă nu se specifică altă poziție (printr-un nod Viewpoint), atunci navigarea începe cu această poziție.

Sunt considerate comentarii toate caracterele după caracterul # până la sfârșitul liniei. Nodurile, câmpurile și valorile câmpurilor pot fi aranjate în oricâte linii; spațiile, tab-urile, virgulele și caracterul de linie nouă sunt considerate spații albe (*white spaces*) și ignorate la citire. VRML este case-senzitive, adică diferențiază literele mici de cele mari.



### 11.3.2 CONSTRUIREA SCENELOR VIRTUALE ÎN VRML

Scenele virtuale se construiesc din noduri de diferite tipuri (noduri de forme geometrice, noduri de grupare, noduri de transformări geometrice, etc.) conectate într-un graf aciclic direcționat.

#### 11.3.2.1 Construirea formelor geometrice

Orice formă geometrică se construiește folosind un nod de tipul *Shape*, cu câmpurile *appearance* și *geometry* pentru specificarea aspectului și a formei geometrice.

Forma geometrică se specifică prin numele unuia din corpurile geometrice elementare recunoscute de VRML (*Box*, *Cylinder*, *Cone*, *Sphere*), printr-o grilă de valori de altitudini (*ElevationGrid*), printr-un obiect obținut prin extrudare (*Extrusion*) sau printr-o rețea de fețe poligonale (*IndexedFaceSet*), linii (*IndexedLineSet*) sau puncte (*PointSet*).

Cea mai puternică și mai folosită modalitate de reprezentare a obiectelor este reprezentarea poligonală descrisă de nodul *IndexedFaceSet*. Acest nod definește un obiect printr-o mulțime de vârfuri (coordonate 3D) și o mulțime de fețe, fiecare față fiind o listă de indici în vectorul de vârfuri. Se poate urmări mai simplu definiția unui astfel de nod în exemplul următor.

#### ■ Exemplul 11.5

```
#VRML V2.0 utf8
# cub.wrl - construiește un cub prin fețe indexate
Shape {
  appearance Appearance {
    material Material { }
  }
  geometry IndexedFaceSet {
    coord Coordinate {
      point [
        # Coordonatele de deasupra a cubului
        -1.0 1.0 1.0,
        1.0 1.0 1.0,
        1.0 1.0 -1.0,
        -1.0 1.0 -1.0,
        # Coordonatele de jos ale cubului
        -1.0 -1.0 1.0,
        1.0 -1.0 1.0,
        1.0 -1.0 -1.0,
        -1.0 -1.0 -1.0
      ]
    }
    coordIndex [
      0, 1, 2, 3, -1,
      7, 6, 5, 4, -1,
      0, 4, 5, 1, -1,
    ]
  }
}
```



```

        1, 5, 6, 2, -1,
        2, 6, 7, 3, -1,
        3, 7, 4, 0
    ]
}
}

```

Nodul Shape creează un cub și specifică fiecare față poligonală printr-o listă de indici în lista de vârfuri. Orice vector (vectorul `point`, vectorul `coordIndex`) este încadrat între paranteze drepte. Delimitarea între indecșii fețelor se face prin numărul `-1`.

### 11.3.2.2 Nodurile de grupare

Nodurile de grupare (Group, LOD, Switch, Billboard, Inline, Collision, Anchor, Transform) ale limbajului VRML sunt noduri care au unul sau mai multe noduri fii.

Nodul **Group** colectează mai mulți fii, care pot fi forme, lumini, sau alte noduri Group. În exemplul următor este creat un obiect compus din trei paralelipiede dreptunghice (semnul "plus" tridimensional).

#### ■ Exemplul 11.6

```

#VRML V2.0 utf8
# semn3D.wrl - semnul "plus" tridimensional
Group {
    children [
        Background {
            skyColor 1.0 1.0 1.0
        },
        Viewpoint {
            position 40.0, 10.0, 40.0
            orientation 0.0 1.0 0.0 0.78
        },
        DirectionalLight {
            direction 0.0 -1.0 -0.2
            intensity 1.0
            ambientIntensity 0.2
            color 1.0 1.0 1.0
        },
        Shape {
            appearance DEF White Appearance {
                material Material ( )
            }
            geometry Box {
                size 25.0 2.0 2.0
            }
        },
        Shape {

```

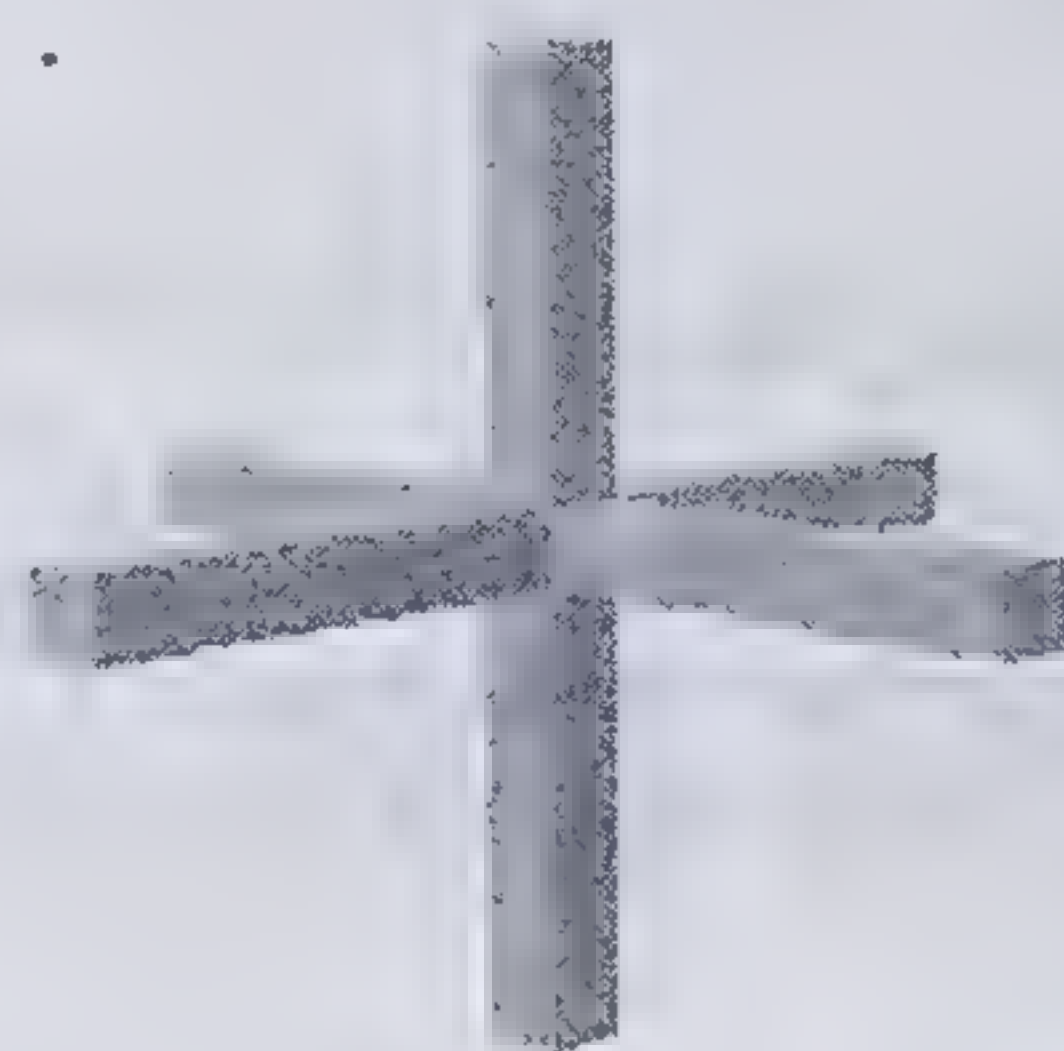


```

        appearance USE White
        geometry Box {
            size 2.0 25.0 2.0
        }
    },
    Shape {
        appearance USE White
        geometry Box {
            size 2.0 2.0 25.0
        }
    }
}

```

Nodul Group conține șase noduri fii: trei noduri de descriere a unei forme geometrice (nodurile Shape), un nod de sursă lumină (DirectionalLight), un nod de definire a culorii de ștergere (Background) și un nod de poziționare a punctului de observare (Viewpoint). Lista de fii este introdusă între paranteze drepte precedate de cuvântul cheie children. În acest exemplu este folosită etichetarea nodurilor. Orice nod poate primi o etichetă introdusă prin cuvântul-cheie DEF. Un nod odată etichetat la definiție, poate fi folosit ca valoare a unui câmp prin simpla introducere a cuvântului-cheie USE urmat de eticheta nodului. În exemplul de mai sus, un nod de aspect a fost etichetat cu numele White (DEF White Appearance{.....}) și apoi folosit cu acest nume ca valoare a unui câmp (appearance USE White). Imaginea care se obține la vizualizarea acestui fișier VRML este dată în fig. 11.10.



**Fig. 11.10** Imaginea afișată de browserul Cosmo Player la încărcarea fișierului semn3D.wrl.

Această imagine se obține fie prin intermediul unui browser (Cosmo Player pentru Netscape), fie prin interpretarea (parsarea) fișierului, crearea grafului scenei și redarea imaginii acestuia.

În fișierul semn3D.wrl mai este folosit un nod pentru definirea culorii de fundal (nodul Background cu câmpul skyColor), nodul Viewpoint pentru definirea poziției de observare și nodul DirectionalLight pentru definirea unei surse de lumină direcțională. Se pot remarca unele diferențe între convențiile



de definire a unor date în VRML și cele din OpenGL. De exemplu, pentru rotație se specifică mai întâi componentele axei de rotație, apoi unghiul de rotație în radiani (în OpenGL se dă mai întâi valoarea unghiului în grade, apoi componentele axei de rotație).

Nodul **Transform** definește transformarea de coordonate a tuturor nodurilor fii ai acestuia față de nodul părinte. Prin aplicarea acestei transformări, un nod fiu este transformat din sistemul de referință propriu în sistemul de referință al nodului părinte. Succesiunea de transformări care se aplică unui nod frunză îl transformă din sistemul de referință local în sistemul de referință universal.

### ■ Exemplul 11.7

Fișierul `cupola.vrl` de mai jos construiește o cupolă din două obiecte poziționate folosind un nod **Transform**:

```
#VRML V2.0 utf8
# cupola.vrl - construiește o cupolă
Group {
  children [
    Background {
      skyColor 1.0 1.0 1.0
    },
    DirectionalLight {
      direction 0.0 -1.0 -1.0
      intensity 1.0
      ambientIntensity 0.2
      color 1.0 1.0 1.0
    },

    # Pereții cupolei
    Shape {
      appearance DEF Brown Appearance {
        material Material {
          diffuseColor 0.6 0.4 0.0
        }
      }
      geometry Cylinder {
        height 2.0
        radius 2.0
      }
    },

    # Acoperișul cupolei
    Transform (
      translation 0.0 2.0 0.0
      children [
        Shape (
          appearance USE Brown
          geometry Cone {
```



de definire a unor date în VRML și cele din OpenGL. De exemplu, pentru rotație se specifică mai întâi componentele axei de rotație, apoi unghiul de rotație în radiani (în OpenGL se dă mai întâi valoarea unghiului în grade, apoi componentele axei de rotație).

**Nodul Transform** definește transformarea de coordonate a tuturor nodurilor fii ai acestuia față de nodul părinte. Prin aplicarea acestei transformări, un nod fiu este transformat din sistemul de referință propriu în sistemul de referință al nodului părinte. Succesiunea de transformări care se aplică unui nod frunză îl transformă din sistemul de referință local în sistemul de referință universal.

### ■ Exemplul 11.7

Fișierul `cupola.vrl` de mai jos construiește o cupolă din două obiecte poziționate folosind un nod **Transform**:

```
#VRML V2.0 utf8
# cupola.wrl - construiește o cupolă
Group {
  children [
    Background {
      skyColor 1.0 1.0 1.0
    },
    DirectionalLight {
      direction 0.0 -1.0 -1.0
      intensity 1.0
      ambientIntensity 0.2
      color 1.0 1.0 1.0
    },

    # Pereții cupolei
    Shape {
      appearance DEF Brown Appearance {
        material Material {
          diffuseColor 0.6 0.4 0.0
        }
      }
      geometry Cylinder {
        height 2.0
        radius 2.0
      }
    },

    # Acoperișul cupolei
    Transform {
      translation 0.0 2.0 0.0
      children [
        Shape {
          appearance USE Brown
          geometry Cone {
```



```

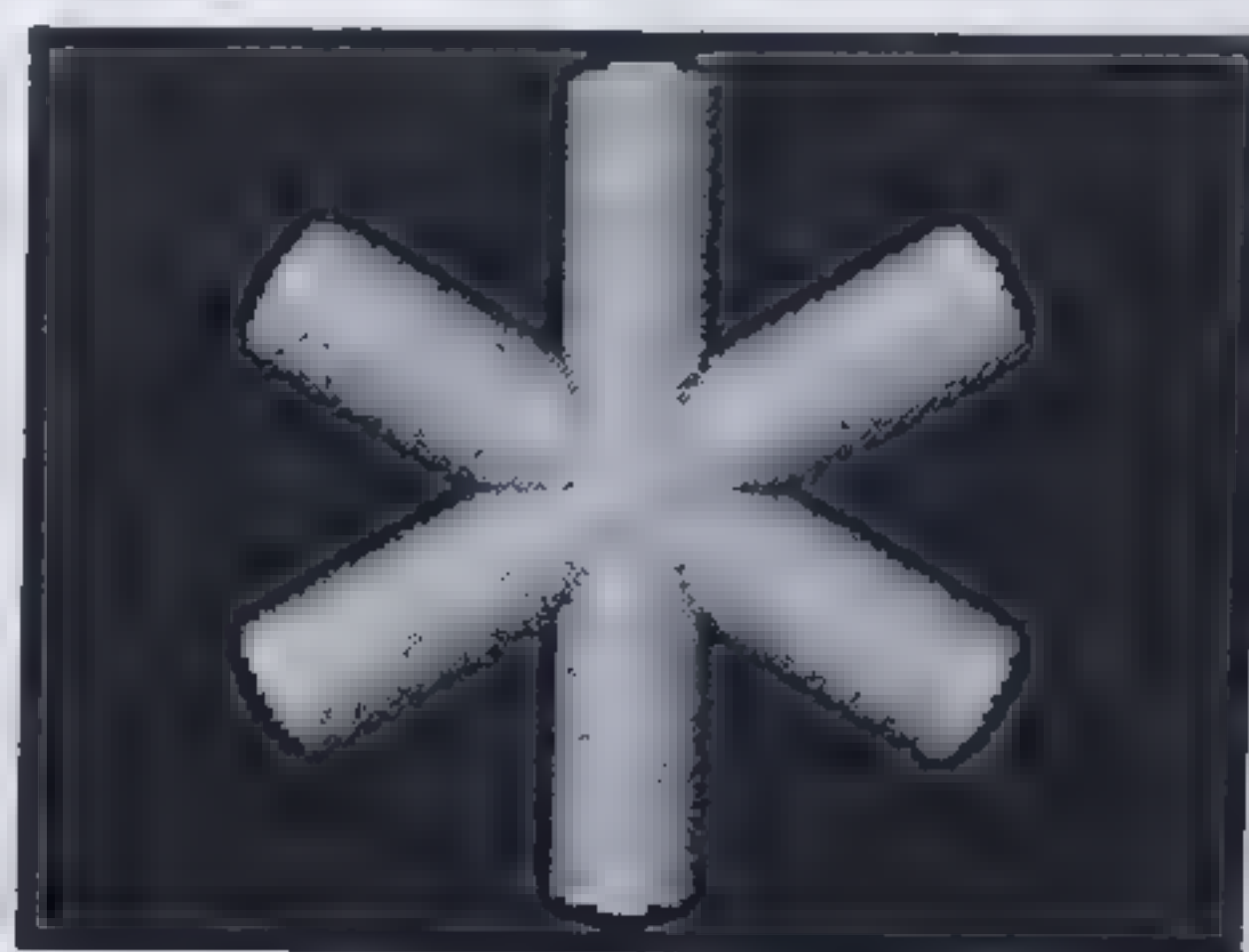
    height 2.0
    bottomRadius 2.5
  }
}
}
}
}
}
}

```

Cupola este reprezentată printr-un nod Group cu doi fii. Primul fiu este un cilindru care redă pereții cupolei; cel de-al doilea fiu este un con translatat cu 2 unități față de nodul părinte (nodul Group) și reprezintă acoperișul. Imaginea obținută prin încărcarea fișierului `cupola.wrl` este dată în fig. 11.11(a).



(a)



(b)

Fig. 11.11 (a) Imaginea fișierului `cupola.wrl`.(b) Imaginea fișierului `asterix.wrl`

### Exemplul 11.8

```

#VRML V2.0 utf8
# asterix.wrl - construiește semnul "asterix" 3D
Group {
  children [
    Viewpoint {
      position 40.0, 10.0, 40.0
      orientation 0.0 1.0 0.0 0.78
    },
    # Primul braț
    DEF Arm1 Shape {
      appearance Appearance {
        material Material { }
      }
      geometry Cylinder {
        height 1.0
        radius 0.1
      }
    },
    # Al doilea braț
    Transform {
      rotation 1.0 0.0 0.0 1.047
    }
  ]
}

```



```

        children USE Arm1
    },
    # Al treilea braț
    Transform {
        rotation 1.0 0.0 0.0 2.094
        children USE Arm1
    }
}

```

Fișierul `asterix.wrl` construiește un semn asterix în trei dimensiuni. Obiectul este reprezentat printr-un nod `Group` cu patru fii. Primul nod fiu (`Viewpoint`) poziționează punctul și direcția de observare. Cel de-al doilea nod fiu este un braț reprezentat printr-un cilindru vertical; cel de-al treilea fiu al nodului `Group` reprezintă al doilea braț și se obține printr-o rotație cu  $60^\circ$  (1.047 radiani) a primului braț relativ la axa x; cel de-al patrulea fiu al nodului `Group` reprezintă al treilea braț și se obține printr-o rotație cu  $120^\circ$  (2.094 radiani) a primului braț relativ la axa x. Grafurile scenelor corespunzătoare fișierelor `cupola.wrl` și `asterix.wrl` sunt date în fig. 11.12.

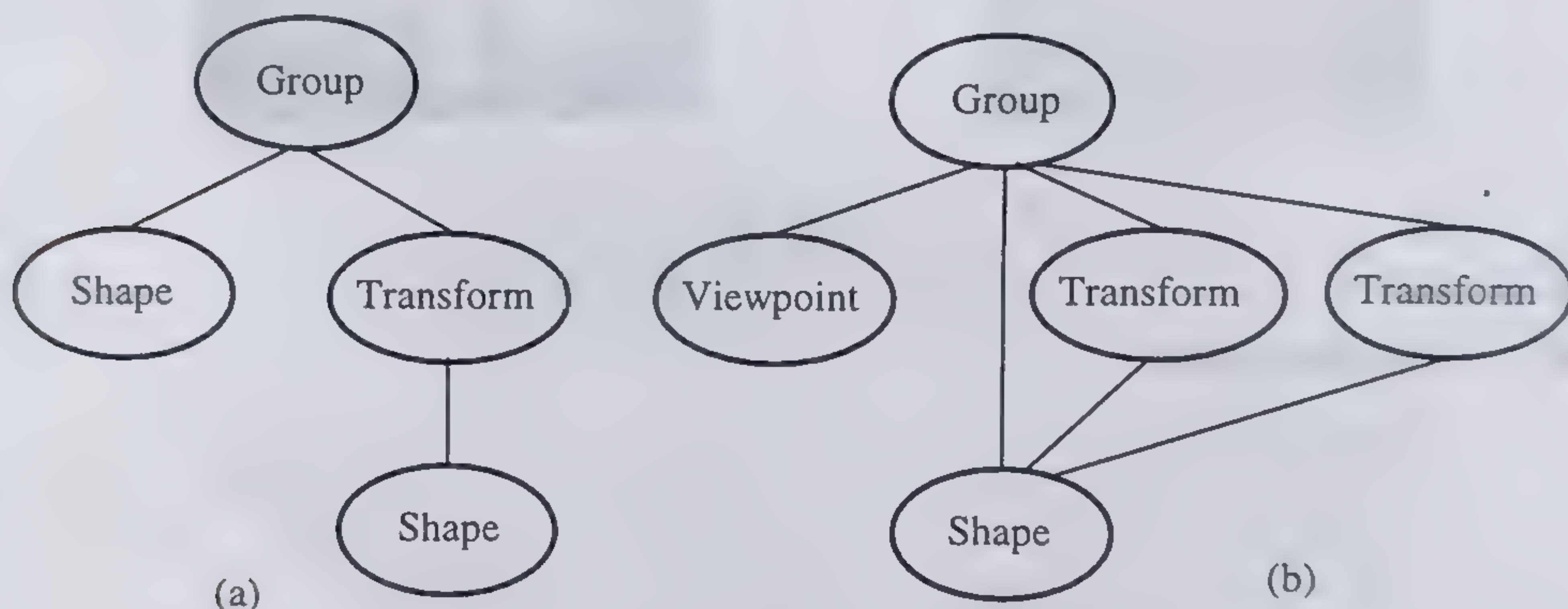


Fig. 11.12 (a) Graful scenei `cupola.wrl` (b) Graful scenei `asterix.wrl`.

**Nodul `Switch`** conține o listă ordonată de noduri fii care descriu reprezentări variate ale aceluiași obiect. În cursul redării, se selectează în secvență câte unul din fii la fiecare cadru de imagine, ceea ce creează impresia de evoluție (modificare dinamică) a obiectului. În acest fel sunt redată luminile de semnalizare (care își schimbă culoarea sau intensitatea la intervale stabilite de timp) sau flăcările, fumul, etc.

**Nodul `LOD`** conține reprezentarea unui obiect cu nivele de detaliu multiple. Fiecare nod fiu al nodului `LOD` este selectat într-un interval de distanțe specificat în vectorul de distanțe al nodului `LOD`.



**Nodul Billboard** definește o transformare a nodurilor fii, dar automat adaugă la această transformare o rotație astfel încât axa z a sistemului de referință al nodurilor fii să fie în permanență orientată către punctul de observare. În acest mod, atunci când observatorul se mișcă în scenă, el vede întotdeauna aceeași față a obiectului. Nodul de tip Billboard se folosește pentru redarea copacilor în scenele virtuale. Un copac se reprezintă printr-o singură față cu o textură care reproduce imaginea copacului și care se rotește astfel încât observatorul vede fața texturată și nu muchiile acesteia.

**Nodul Inline** este un nod de grupare care permite introducerea unui sau mai multor noduri fii dintr-un alt fișier VRML extern specificat prin codul URL. În acest fel, o scenă virtuală se poate compune din mai multe module (fișiere) existente.

### 11.3.3 ANIMAȚIA ÎN VRML

Una din cele mai importante caracteristici ale limbajului VRML este abilitatea de a descrie animația obiectelor în scena virtuală. În VRML se pot modifica multe caracteristici ale scenei virtuale, ca de exemplu:

- poziția, orientarea și dimensiunea obiectelor;
- culoarea, materialul, textura și parametrii de aplicație a texturii pe fețele obiectelor;
- culoarea, poziția și direcția luminilor.

Caracteristica animației în VRML este aceea că fiecare nod al scenei este tratat ca un element cu intrări și ieșiri, iar prin conectarea acestora se obține un “circuit cablat” care poate fi utilizat ca un traseu de animație, de-a lungul căruia se pot deplasa date în scenă. În limbajul VRML, o conexiune între două noduri din scenă se numește traseu (*route*), o dată care se deplasează se numește eveniment (*event*), iar intrările și ieșirile în noduri se numesc evenimente de intrare (*eventsIns*), respectiv, evenimente de ieșire (*eventsOuts*).

De exemplu, dacă un nod are un câmp numit *rotation*, atunci nodul respectiv are un eveniment de intrare numit *set\_rotation* și un eveniment de ieșire numit *changed\_rotation*.

Pentru crearea unui traseu de animație, se definesc noduri de transformare și noduri de interpolare, care asigură o interpolare liniară a valorilor de deplasare, rotație sau modificare a culorilor.

În VRML sunt definite șase tipuri de noduri de interpolare: *PositionInterpolator*, *OrientationInterpolator*, *ScalarInterpolator*, *CoordinateInterpolator*, *NormalInterpolator*, *ColorInterpolator*. Prin conectarea nodurilor de transformare și de interpolare (folosind comanda *ROUTE*), se obține o mare varietate de animație a obiectelor în scenă.



### 11.3.4 CARE ESTE VIITORUL LIMBAJULUI VRML?

În momentul de față (anul 2000) se pregătește o nouă versiune a limbajului VRML, numită, deocamdată, VRML NG și, de asemenea, noi toolkit-uri de proiectare a scenelor tridimensionale. De exemplu, Silicon Graphics și Microsoft au anunțat toolkit-ul Farenheit, iar Sun a dezvoltat toolkit-ul Java3D pentru crearea aplicațiilor 3D folosind limbajul Java.

La fel ca în VRML, Java3D permite reprezentarea scenelor 3D complexe printr-un graf al scenei. Nodurile fii ale grafului sunt forme geometrice, materiale, lumini, sunete, etc, iar nodurile părinte sunt noduri de transformare sau de grupare. De fapt, pentru un cunoscător al limbajului VRML, terminologia din Java3D este foarte familiară. Cele mai multe noduri din Java3D au funcționalități similare celor din VRML, dar, în plus, sunt prevăzute funcționalități suplimentare, cum ar fi combinarea culorilor prin transparență, definirea modului de filtrare a texturilor, inclusiv filtrarea mip-map, efectul ceții, etc.

Versiunea 1.1 a toolkit-ului Java3D este deja disponibilă (gratuit) atât pentru sisteme Sun Solaris, cât și pentru sisteme Windows NT sau 95. Specificații Java3D se pot găsi la adresele <http://java.sun.com/products/java-media/3D/>, <http://www.sun.com/deskop/java3d/> sau <http://java3d.sdsc.edu>.

Se poate remarca deci o tendință de unificare a reprezentării scenelor virtuale pentru diferite platforme de calcul, sisteme de operare, limbaje de programare și toolkit-uri de dezvoltare pe baza specificațiilor limbajului VRML.

## 11.4 APLICAȚII ALE REALITĂȚII VIRTUALE

Aplicații ale realității virtuale se întâlnesc în numeroase domenii de activitate, bazate pe diferite sisteme hardware/software de dezvoltare [Bry95]. Unul din cele mai importante domenii de activitate în care se folosesc sisteme de realitate virtuală este cel al aplicațiilor militare și aerospațiale, în care astfel de sisteme (*simulatoarele de antrenament*) reduc considerabil atât costurile de antrenament, cât și pericolul de exersare a manevrelor dificile. Un alt domeniu de activitate, cu un număr mai mic de aplicații decât cel din domeniul aerospațial, dar cu un rol deosebit de important în evitarea și reducerea pierderilor de vieți omenești, este medicina, în special chirurgia. Alte domenii în care sistemele de realitate virtuală au un rol important sunt: educație, artă, afaceri, proiectare, robotică, divertisment, cercetare [Burd94], [Earn93], [Earn95], [Slat99]. În continuarea acestui capitol vor fi prezentate pe scurt o parte din cele mai interesante aplicații ale realității virtuale.

### 11.4.1 SIMULATOARE DE ANTRENAMENT

Simulatoarele de antrenament se construiesc pentru formarea deprinderilor de manevrare a unor vehicule: aerospațiale, navale, rutiere, sau de cale ferată.



Dintre acestea, cea mai mare dezvoltare au cunoscut-o simulatoarele din industria aeronautică și aerospațială. Costul deosebit de mare al aparatelor de zbor, dotarea acestora cu echipamente complexe de navigare sau luptă aeriană, cerințele de efectuare a unor manevre complicate, cu mare viteză și precizie, au impus necesitatea pregătirii personalului de zbor înainte de a porni în misiune.

Un simulator de zbor este, de asemenea, o instalație complicată care utilizează o mare varietate de tehnologii și echipamente (echipamente hidraulice, electronice, sisteme de calcul în timp real, proiectoare optice, generatoare de imagini 3D, etc.) și este cel mai potrivit loc unde pilotul se poate acomoda cu cabina aparatului real [Vin93]. Acest lucru este posibil deoarece cabina simulatorului reproduce cu fidelitate un aparat specific (de exemplu: Airbus 320, Concorde, Boeing 767, Mig 21, etc.). Dintre avantajele oferite de antrenarea pe simulatoare se pot aminti:

- Securitatea completă a instruirii, pentru pilot și aparat, atât în condiții normale cât și în condiții dificile și periculoase (defectări ale echipamentelor aeronavei, condiții meteorologice grele, etc.).
- Economie de combustibil și evitarea uzurii aparatelor de zbor.
- Evaluarea obiectivă a performanțelor echipajului prin analiza evenimentelor de zbor simulat.

Simulatoarele de antrenament sunt sisteme de realitate virtuală semi-immersive, în care se combină echipamente reale (elementele de comandă din cabină) cu imaginea mediului virtual, cu sunetului virtual și cu senzația de mișcare generată artificial. Pilotul este ambarcat în scaunul fixat rigid de cabină, iar scenariile de antrenament îl obligă să poarte centură în fazele de decolare și aterizare. Cea mai puternică senzație de imersiune în mediul virtual este asigurată prin imaginea vizuală, care trebuie să reproducă condițiile de mediu cât mai realist posibil: aeroportul cu pista și clădirile înconjurătoare, terenul cu reperele naturale sau construite (râuri, păduri, șosele, poduri, orașe, etc.) [Ion98c]. Toate aceste date constituie baza de date grafice, generată off-line, care este încărcată în memoria generatorului de imagine și traversată în timp real în cursul antrenamentului. Calitatea imaginii generate, ca realism, rezoluție, complexitate și frecvență de actualizare, reprezintă un factor important în instruirea corespunzătoare a piloților. Datorită cerințelor de putere de calcul deosebit de ridicată, generatoarele de imagine vizuală au fost realizate, în general, ca sisteme de *procesare paralelă și distribuită*, în diferite arhitecturi special proiectate sau folosind stații grafice.

Scenariile de antrenament ale piloților conțin, de regulă, exersarea manevrelor normale (decolare, aterizare, zbor de croazieră), ca și a manevrelor care se impun în situații de avarii ale aparatului. Răspunsul pilotului pentru fiecare situație specială este comparat cu cerințele bine precizate pentru aparatul respectiv și introduse ca parte componentă a modelului aparatului. Manevrele greșite sunt identificate și penalizate corespunzător. Din fericire, în simulator, o manevră greșită poate fi reluată ori de câte ori este nevoie pentru deprinderea ei corectă, ceea ce în zborul real nu este posibil, orice manevră greșită putând conduce la o catastrofă. Succesul antrenării poate fi apreciat prin evaluarea îndemânării obținute



și prin capacitatea pilotului de a efectua un zbor real, după antrenarea pe un simulator de zbor.

*Din punct de vedere funcțional*, un simulator de zbor se compune din mai multe sisteme intercorelate: sistemul de modelare a aeronavei; sistemul de comandă a aparatelor de bord (avionica), sistemul vizual (generatorul de imagine), sistemul sonor (generatorul de sunet), sistemul de mișcare, sistemul de efort în comenzi. Fiecare sistem al simulatorului este compus dintr-un modul de comandă, interfețe specifice și dispozitive de intrare-ieșire.

*Din punct de vedere constructiv*, un simulator de zbor este compus din două subansamble: subansamblul de pilotaj și subansamblul de comandă și control, interconectate printr-o magistrală de date.

Subansamblul de pilotaj este alcătuit din cabina de pilotaj, amplasată pe o platformă care îi asigură mișcările necesare. Cabina este echipată cu toate comenzile de zbor corespunzătoare comenzilor reale și cu toate aparatele de bord indicatoare, cu aspect și funcționare cât mai apropiată de cea a aparatelor reale. În fața cabinei sunt amplasate dispozitivele de redare a imaginii mediului virtual (monitoare sau ecran și proiectoare), precum și difuzoarele pentru generarea sunetului virtual.

Subansamblul de comandă și control este compus din stațiile de calcul conectate în rețea locală, care comandă atât cabina de pilotaj cât și un post al instructorului conținând aparate de urmărire a zborului. Aparatele de urmărire a zborului din postul instructor nu mai sunt, în mod necesar, identice ca aspect cu cele din cabina reală, și sunt, în general, aparate virtuale, generate pe unul sau mai multe displayuri conectate la stațiile din rețea. În plus, sunt prevăzute numeroase facilități de urmărire a traiectului de zbor, a anvelopelor de zbor (valorile limită ale parametrilor) precum și redarea imaginii mediului virtual care dublează imaginea generată în cabină.

*Din punct de vedere arhitectural*, un simulator de zbor este compus dintr-un calculator central (stație), care este conectat în rețea cu un număr de alte stații în care sunt amplasate modulele de comandă ale diferitelor sisteme ale simulatorului.

Sistemul central al unui simulator este *sistemul de modelare a aeronavei*, care este executat pe stația centrală a simulatorului (consola sistem). Datele de intrare ale modelului sunt comenzile date de pilot prin acționarea echipamentelor din cabină, preluate de interfețele specifice și transmise prin magistrala de date. Pe baza acestor comenzi are loc integrarea (în timp real) a datelor dinamice ale aeronavei, ale motoarelor de propulsie, ale echipamentelor electrice, hidraulice, navigație, armament, etc. În fiecare pas de iterație, pe baza mărimilor de stare și a mărimilor de intrare, se generează comenzile pentru toate celelalte sisteme funcționale ale simulatorului.

Celelalte sisteme ale unui simulator (sistemul de comandă a aparatelor de bord/avionică, sistemul de imagine, sistemul de sunet, sistemul de mișcare, sistemul de efort în comenzi) prelucrează comenzile primite de la sistemul de modelare și prezintă datele rezultate în urma prelucrării în mod specific fiecărui sistem: indicațiile aparatelor de bord, imaginea scenei virtuale proiectate pe ecranul



din cabină, sunetul generat în difuzoare, mișcarea platformei pe care este amplasată cabina, senzația tactilă de efort la acționarea manetelor sau pedalelor).

Pilotul acționează în orice moment în funcție de aceste date. Se poate spune că un simulator este un sistem în buclă închisă, închiderea buclei fiind asigurată de însuși pilotul (fig. 11.13).

Conexiunile reprezentate în fig. 11.13 între sistemele componente ale unui simulator sunt conexiuni logice. În realitate, toate stațiile de calcul sunt conectate în rețea locală printr-un segment Ethernet prin care se transferă mesaje punct-la-punct sau mesaje de difuziune (broadcast). Magistrala de date care conectează subansamblul cabină de subansamblul de comandă și control folosește, în general, transmisia multiplexată a datelor.

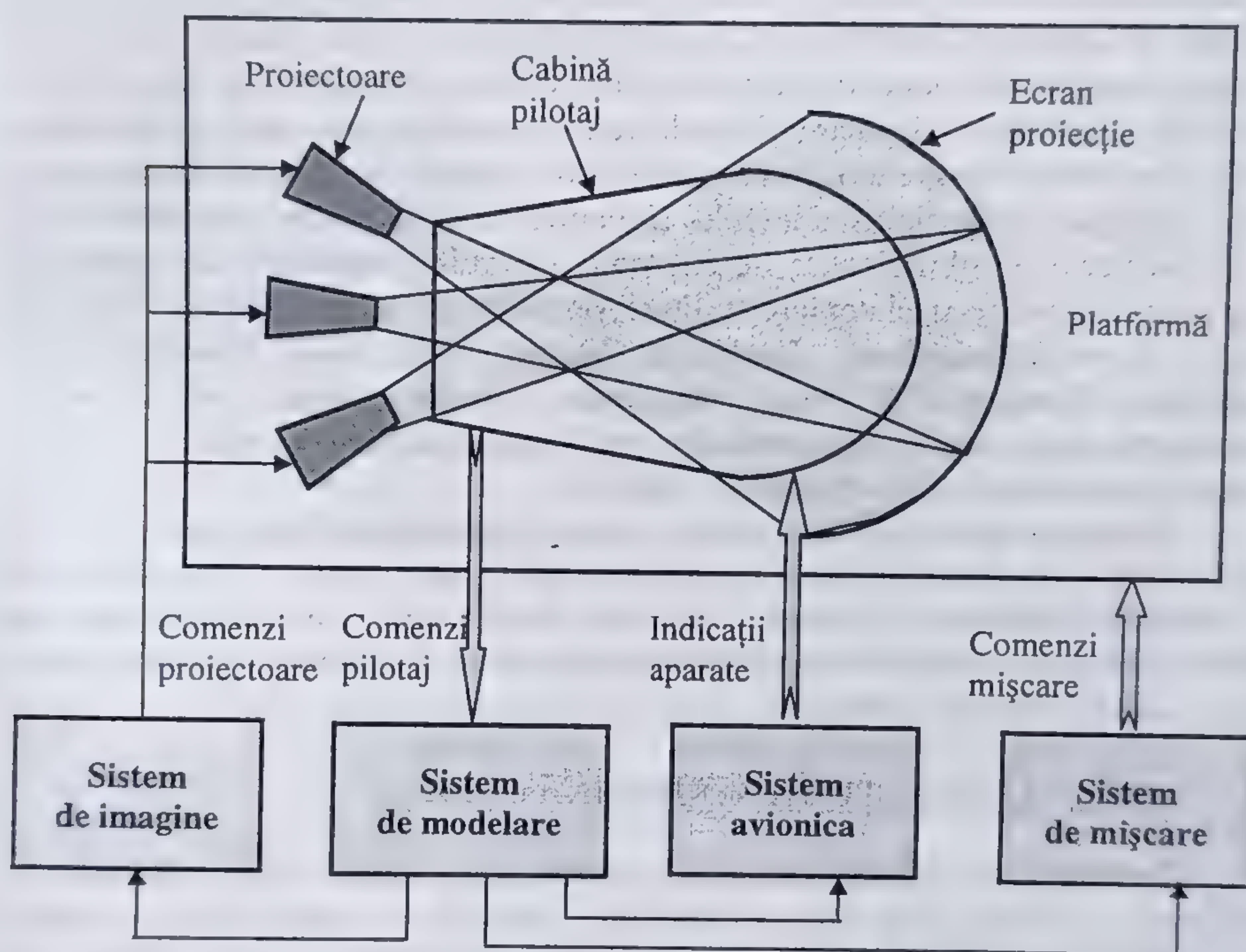


Fig. 11.12 Sistemele componente ale unui simulator de zbor.

Pentru o simulare de calitate sunt importante:

- Fidelitatea modelării.
- Fidelitatea redării zborului, așa cum este perceput de către pilot (aparate de bord, scena vizuală, sunet, mișcare).
- Rată suficientă de mare de iterare a calculelor (pentru a crea impresia de continuitate).
- Timp de răspuns cât mai mic (intervalul de timp dintre momentul unei acțiuni a pilotului și efectul acestei acțiuni, prezentat prin intermediul imaginii, indicațiilor aparatelor de zbor, etc.).



Una din senzațiile cele mai dificile de simulat este senzația de mișcare. În timpul zborului real, pilotul percepe mișcarea integrând informații provenind de la imaginea scenei exterioare (deplasarea orizontului și a obiectelor în câmpul vizual), de la indicațiile aparatelor de bord (indicații de altitudine, viteză, orientare), senzații tactile (percepute de senzorii musculari), senzații de mișcare (percepute de sistemul vestibular al urechii).

Într-un simulator de zbor, senzația de mișcare se generează prin reprezentarea corespunzătoare a imaginilor (imaginea în mișcare a mediului virtual și indicațiile aparatelor de bord) și prin accelerarea cabinei simulatorului folosind platforme (în general cu 6 grade de libertate) acționate de sisteme hidraulice puternice. Cele două surse de senzații se completează reciproc, creând o percepție consolidată a mișcării [Botea97]. Senzațiile de mișcare trebuie generate cu maximă fidelitate în simulator, atât pentru consolidarea percepției generale a zborului, cât mai ales atunci când manevra exersată cere precizie a comenziilor: zbor la joasă altitudine, manevre de decolare și aterizare, zbor în formație, zbor instrumental, zbor în condiții meteo grele. Lipsa senzațiilor de mișcare sau proasta coordonare a acestora produce imprecizia manevrelor și manifestarea "răului de simulator".

În perioada ultimilor 25 de ani s-au construit numeroase simulatoare de zbor, pentru avioane și elicoptere militare și civile, de mari firme din SUA sau Europa (Evens&Sutherland, Lockheed Martin, McDonnell Douglas, CAE Electronics, Thomson CSF, etc.). Tehnologia și performanțele simulatoarele de antrenament au evoluat și evoluează în continuare cu rapiditate, astfel încât orice prezentare într-o lucrare devine repede depășită.

În țara noastră s-au construit mai multe simulatoare militare pentru avioane și elicoptere, la Institutul de Simulatoare Simultec S.A., care se află în exploatare în Centrul de Instruire a Piloților, Măgurele. În Planșa 10 este redată imaginea generată în timpul zborului într-un simulator de avion. În imagine se vede terenul, copaci, clădiri, drumuri, vehicule.

#### 11.4.2 APLICAȚII ÎN MEDICINĂ

Introducerea calculatoarelor în medicină a îmbunătățit metodele de pregătire a medicilor și de îngrijire medicală. Sisteme de calcul puternice permit menținerea și consultarea unor informații utile sub forma de baze de date ale pacienților sau ale diagnosticelor, consultații la distanță, radiografii digitale, simulări prechirurgicale.

Utilizarea realității virtuale în chirurgie are un impact puternic în ceea ce privește antrenarea chirurgilor în practicarea unor noi proceduri chirurgicale, în planificarea operațiilor complexe sau în prezentarea informațiilor de "navigare" în cursul unor intervenții chirurgicale.

Formarea unui chirurg cu adevărat competent în operații complicate necesită efectuarea unui număr foarte mare de operații, în mulți ani după absolvirea studiilor. Exersarea operației pe cadavre este foarte dificilă deoarece, după lezarea organului investigat, operația nu mai poate fi repetată. Posibilitatea ca un chirurg să repete o procedură chirurgicală, până la perfecțiune, înainte de a o efectua asupra



pacientului, permite scăderea riscului la care este supus pacientul. În sistemele de realitate virtuală de *simulare a operațiilor chirurgicale*, este importantă generarea unei imagini cât mai realiste a organelor investigate, precum și simularea senzației tactile și a forței de reacție, prin folosirea unei mănuși de date.

Un astfel de simulator chirurgical pentru intervenții abdominale a fost dezvoltat de firma Division și implementat pe două stații Silicon Graphics IRIS 310 VVG, care afișează imaginea stereoscopică pe display-urile unei căști de vizualizare HMD EyePhone. Sistemul este prevăzut de asemenea cu două mănuși de date DataGlove [Sat92].

Una din cele mai delicate tehnici chirurgicale, *tehnica cu invaziune minimală* (*minimally invasive surgery*), necesită o antrenare intensă a chirurgului, datorită condițiilor în care acesta trebuie să opereze. În astfel de intervenții (cum sunt laparoscopia și endoscopia), o minicameră și un instrument de incizie sunt introduse în corpul pacientului. Chirurgul acționează asupra instrumentului de incizie de la distanță, privind imaginea redată pe un monitor TV. Pentru reușita operației, este necesară o coordonare perfectă între imaginea văzută pe display și mișcarea mâinii, coordonare care se obține numai printr-o antrenare intensă și care nici nu poate fi exersată pe cadavre. Numărul foarte mare de intervenții cu invaziune minimală (apreciat la mai multe milioane pe an) a produs o adevărată explozie în realizarea și comercializarea *simulatorilor de intervenții cu invaziune minimală*. În general, astfel de simulatoare sunt imersive, generând o imagine stereoscopică pe display-urile unui HMD și posedă obligatoriu o mănușă de date, pentru exersarea sincronizării între imaginea văzută și mișcarea mâinii.

O altă aplicație importantă din domeniul medicinei îl reprezintă *simulatorul de anatomie*. Metoda tradițională de învățare a anatomiei, folosind fotografii bidimensionale sau disecția cadavrelor, poate fi înlocuită astăzi cu studierea unui model anatomic al corpului uman. Astfel de modele au fost dezvoltate în mai multe variante, pornind de la date provenite din disecții și pot fi achiziționate sau accesate la distanță în spitale sau universități. O simulare de calitate a *corpului virtual* se obține prin imersiune, folosind un sistem de vizualizare cu cască montată pe cap (HMD) și o mănușă de date.

Un sistem de realitate virtuală poate constitui atât un instrument de instruire, cât și de experimentare [Sat95]. De exemplu, un student poate urmări un traseu într-un anumit organ sau sistem anatomic, după care poate experimenta o anumită intervenție în acesta. Astfel de sisteme se pot considera sisteme educaționale 4D: la spațiul tridimensional se adaugă a patra dimensiune, timpul; studentul "merge" prin stomac și "vede" un ulcer, ceea ce permite o acumulare de cunoștințe incomparabilă cu aceea obținută din planșe sau disecții.

În medicină se mai întâlnesc numeroase alte aplicații ale realității virtuale: în telechirurgie, diagnosticare, recuperare, terapie [Nor98], [Lear97]. De exemplu, sisteme de realitate virtuale pot fi utilizate pentru tratarea fricii, a fobiilor sau a stresului, dar această utilizare poate fi riscantă și este supusă unor reglementări medicale internaționale.



### 11.4.3 APLICAȚII ÎN ARTĂ ȘI DIVERTISMENT

Divertismul reprezintă cea mai mare piață pentru produsele de realitate virtuală și numeroase firme cu mare pondere financiară s-au implicat în astfel de aplicații. Există o mare varietate de forme de divertisment în medii virtuale, de la jocuri video la domiciliu, până la sisteme complexe cu imersiune.

De asemenea, au fost organizate numeroase expoziții de jocuri în realitate virtuală, în care se desfășoară competiții între echipe de joc, prin intermediul unei rețele de comunicație care asigură o simulare interactivă distribuită. Caracteristica importantă a unor astfel de manifestări (cum au fost cele cu simulatoarele *Battle Tech* sau *Mirage*) este aceea că, deși este vorba de divertisment, nivelul tehnic este deosebit de ridicat, atingând nivelul simulatoarelor militare și asigurând o puternică senzație de realism. Astfel de expoziții au atras numeroși vizitatori și, bineînțeles, venituri importante.

Pe aceleași principii au fost create numeroase filme de science-fiction și alte producții artistice, realitatea virtuală reprezentând un nou mijloc de expresie artistică. Tendința care se evidențiază în producțiile artistice prin intermediul realității virtuale este aceea de a crea o interacțiune spectator-calculator-artist sau chiar direct între spectatori, ceea ce dă produselor artistice (picturi, compoziții muzicale, piese de teatru) un caracter dinamic, în continuă schimbare. Această caracteristică diferențiază puternic arta virtuală de arta clasică, așa cum este prezentată în muzee sau expoziții.

O idee fascinantă de utilizare a realității virtuale în artă o constituie *muzeele virtuale*. Orice posesor al unui calculator ar putea "vizita" oricare muzeu de pe glob, fără să se deplaseze de la propriul domiciliu. Bineînțeles, vizitarea muzeului virtual nu poate înlocui vizitarea muzeului real, decât în măsura în care sistemul grafic pe care se execută vizionarea este deosebit de performant.

Un alt experiment interesant îl constituie *teatrul virtual*. În anul 1998 un grup de cercetători canadieni au inițiat un proiect ambițios: montarea piesei lui Shakespeare, *Visul unei nopți de vară*, și difuzarea acesteia pe Internet (sub numele de VRML Dream) [Mat99]. Decorul și actorii au fost modelați în VRML 2.0, iar vocile au fost oferite de un grup de actori (reali), digitizate și comprimate. În mai puțin de un an, VRML Dream a cunoscut o mare audiență pe Internet, unde poate fi accesat printr-un browser (ca, de exemplu, CosmoPlayer) la adresa <http://www.vrmldream.com/vrmldream/artistic.html>.

### 11.4.4 FACTORUL UMAN, ETIC ȘI ESTETIC ÎN REALITATEA VIRTUALĂ

Realitatea virtuală a devenit binecunoscută și mediatizată în ultimii ani, dar, de multe ori, speculațiile au creat mituri despre ceea ce este sau ce ar putea realiza aceasta, mituri care nu corespund întotdeauna cu starea de fapt. Realitatea virtuală a creat o nouă formă de interacțiune om-calculator, cu o interfață fundamental nouă. În locul utilizării în mod convențional a tastaturii și ecranului,



utilizatorul poartă o cască de vizualizare pe cap, iar o mănușă de date permite interacțiunea cu mediul virtual.

Esența realității virtuale este imersiunea, participanții având senzația că fac parte din mediul virtual. Sistemele de realitate virtuală se află într-o evoluție extrem de rapidă, în care dezvoltările hardware și software au loc în mod permanent și sunt accesibile pentru un număr din ce în ce mai mare de utilizatori. În consecință, studierea realității virtuale din punct de vedere al factorului uman, etic și estetic, precum și al posibilelor efecte colaterale asupra utilizatorilor, a devenit o necesitate.

**Factorul uman.** "Răul de mișcare" (*motion sickness*) este binecunoscut și documentat. Este vorba atât de "răul de mare", resimțit în călătoriile cu vaporul, cât și de "răul de zbor", resimțit în aeronave sau nave spațiale. O altă formă de "rău de mișcare" este "răul de simulator", raportat de utilizatorii simulatoarelor de antrenament încă din anul 1957. Acesta constă din numeroase modificări funcționale la nivel gastric, hormonal, cardiovascular și respirator, cu manifestări de amețeală, greață, paloare, tremurat, etc.

Numeroase studii efectuate asupra "răului de simulator", pe mii de simulatoare și piloți antrenați, au stabilit că, în majoritatea cazurilor, în timp, are loc o acomodare și o scădere a simptomelor adverse. Există însă și cazuri în care acomodarea poate să nu aibă loc niciodată. De asemenea, se cunosc și simptome adverse care persistă, mai mult sau mai puțin timp, după participarea într-un simulator. Considerându-se că "răul de simulator" este o problemă serioasă, care poate reduce puternic eficiența antrenării, s-au dezvoltat mai multe teorii asupra cauzelor care îl provoacă. Majoritatea acestora au stabilit că simptomele adverse sunt cauzate de un *conflict senzorial* și apar datorită neconcordanței între stimulii vizuali și cei vestibulari sau între stimulii primiți și ceea ce persoana se așteaptă să simtă, conform experiențelor anterioare [Reg95].

Realitatea virtuală imersivă implică un conflict senzorial datorită neconcordanței între stimulii vizuali și stimulii vestibulari. În cele mai multe sisteme de realitate virtuală imersive, mișcarea mâinii (echipată cu un dispozitiv de interacțiune) este cea care provoacă modificarea imaginii, în timp ce poziția corpului (care controlează sistemul vestibular) rămîne neschimbată. La acest conflict se mai adaugă conflictul datorat întârzierii de răspuns a sistemului, ceea ce provoacă neconcordanță de timp între mișcarea fizică reală, care acționează ca stimuli vestibulari, și modificarea imaginii, care acționează ca stimuli vizuali.

Rezultate experimentale, efectuate pe numeroase sisteme de realitate virtuală și numeroase echipe de utilizatori [Ken89], au confirmat existența simptomelor adverse și amenințarea pe care acestea o reprezintă pentru extinderea utilizării sistemelor de realitate virtuală. De aceea, studierea în continuare a cauzelor "răului de simulator", pentru a putea fi eliminat sau redus, este de o mare importanță pentru viitorul sistemelor de realitate virtuală.

**Factorul etic.** Experimentele de terapie psihiatrică prin intermediul realității virtuale trebuie privite cu circumspecție, cu atât mai mult cu cât progresul



tehnologic le fac accesibile unui număr mare de experimentatori entuziaști, dar lipsiți de cunoștințe științifice temeinice.

De aceea, în privința utilizării realității virtuale în recuperare și terapie, s-a conturat necesitatea unor reglementări de etică medicală, care să împiedice abuzul sau experimentele pseudo-științifice, cu posibile efecte nocive asupra pacienților [Wha93]. Astfel de reglementări au apărut deja, atât în SUA cât și în Europa, și ele statuează principiile unei *practici clinice corecte* (*Good Clinical Practice*, 1990). Aceste principii includ obligativitatea obținerii acordului subiectului de a fi supus cercetărilor sau experimentelor medicale (inclusiv a celor bazate pe sisteme de realitate virtuală).

**Factorul estetic.** Rolul educativ al multor expoziții sau muzee virtuale este unanim recunoscut, dar, de multe ori, experimentele oferite sunt lipsite de profunzime estetică sau culturală. De aceea, se încearcă implicarea artiștilor contemporani în crearea galeriilor de artă virtuală, pentru a asigura nivelul estetic al artei construite prin intermediul realității virtuale.

O astfel de implicare a fost încercată în anul 1994, când a fost lansat proiectul *Galeria Virtual* [Par95], la care au participat numeroși artiști, invitați să-și exprime propria viziune asupra unei "realități" care extinde realitatea cunoscută. Rezultatele obținute în *Galeria Virtual*, ca și în alte manifestări de acest gen, conturează factorul creativ și estetic al producțiilor artistice bazate pe realitatea virtuală.



## BIBLIOGRAFIE

- [Adam93] J. Adam, "Virtual Reality is for Real", IEEE Spectrum, IEEE Press, October, 1993.
- [Ames97] A.L. Ames, D.R. Nadeau, și J.L. Moreland, *The VRML 2.0 Sourcebook*, John Wiley & Sons Inc., New York, 1997.
- [Aur91] F. Aurenhammer, "Voronoi Diagrams-A Survey of Fundamental Geometric Data Structure", ACM Computing Survey, Vol. 23 Nr. 3, Sept. 1991.
- [Azu97] R.T. Azuma, "A survey of Augmented Reality", Presence: Teleoperators and Virtual Environments, Vol. 6, Nr. 4, August 1997, pp. 355-385.
- [Baciu99] Rodica Baciu și D. Volovici, *Sisteme de prelucrare grafică*, Editura Albastră, Cluj-Napoca, 1999.
- [Bez72] P. Bezier, "Numerical Control: Mathematics and Applications", Chichester-Wiley, 1972.
- [Bla98] E. Blake, *Interactive Computer Graphics*, online, <http://www.cs.utc.ac.za/Courses/CS300WICG/>
- [Bier86] E.A. Bier și K.R. Sloan, "Two-part texture mapping", IEEE Computer Graphics and Applications, Vol. 6, Nr. 9, 1986, pp. 40-5.
- [Botea97] C. Botea, *Comanda mișcării simulatoarelor de zbor*, teză de doctorat, Universitatea Politehnica București, 1997.
- [Bre65] J.E. Bresenham, "Algorithm for computer control of digital plotter", IBM Systems Journal, January 1965, pp. 25-30.
- [Brown95] D.J. Brown, S.V. Cobb și R.M. Eastgate, "Learning in virtual environments", R.A. Earnshaw (ed), *Virtual Reality Systems*, Academic Press, London, 1995.
- [Bry95] S. Bryson, "Approaches to the succesful design and implementation of VR applications", R.A. Earnshaw (ed), *Virtual Reality Applications*, Academic Press, London 1995.
- [Bucur97] C. Bucur și Felicia Ionescu, "Debriefing Software as a Tool for Airline Operation Efficiency Improvement", in Proceedings of the 8<sup>th</sup> International Training and Education Conference ITEC'97, pp. 199-203, April 1997, Lausanne, Switzerland.
- [Burd92] G. Burdea, J. Zhuang, E. Roskos, D. Silver și N. Langrana, "A Portable Dextrous Master with Force Feedback", Presence- Teleoperators and Virtual Environments, Vol. 1, Nr. 1, pp 18/27, March 1992.
- [Burd93] G. Burdea, "Virtual Reality Systems and Applications", Electro'93 International Conference, Short Course, Edison, NJ, April 1993.
- [Burd94] G. Burdea și P. Coiffet, *La Réalité Virtuelle*, Edition Hermes, Paris, 1994.
- [Burd97] G. Burdea, *Force and Touch Feedback for Virtual Reality*, John Willey & Sons, Inc., New York, 1997.
- [Car97] M.S.T. Carpendale, D.J. Cowperthwaite și D. Fracchia, "Extended Distorsion Viewing from 2D to 3D", IEEE Computer Graphics and Applications, Vol. 17, No. 4, 1997.
- [Car84] L.C. Carpenter, "The A-buffer, an anti-aliasing hidden surface method", Computer Graphics, Vol. 18, Nr. 3, 1984, pp. 103-108.
- [Car99] G.S. Carson, R.F. Puk și R. Carey, "Developing the VRML 97 International Standard", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.
- [Cat75] E. Catmull, "Computer display of curved surfaces", Proc. IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures, May 1975.
- [Cat78] E. Catmull, "A hidden surface algorithm with anti-aliasing", Computer Graphics, Vol. 12, Nr. 3, 1978, pp. 6-10.



- [Chau97] J.C. Chauvin, "Geographical Coordinates in Real-Time Image Generation", Proceedings of the 8<sup>th</sup> International Training and Education Conference ITEC'97, April 1997, Lausanne, Switzerland.
- [Cohen97] D. Cohen-Or și A. Kaufman, "3D Line Voxelization and Connectivity Control", IEEE Computer Graphics and Applications, Vol. 17, No. 6, Nov/Dec. 1997.
- [Craw81] F.C. Craw, "A comparison of anti-aliasing techniques" IEEE Computer Graphics and Applications, Vol. 1, Nr. 1, pp. 40-48, 1981.
- [Cig97] P. Cignoni, E. Puppo și R. Scopigno, "Multiresolution Representation and Visualization of Volume Data", IEEE Transaction on Visualization and Computer Graphics, Vol. 3, No. 4, October-December 1997.
- [Del97] C. Delepine, "Online terrain level of detail", Proceedings of the 8<sup>th</sup> International Training and Education Conference ITEC'97, pp. 23-28, April 1997, Lausanne, Switzerland.
- [Dev96] V. Devarajan, R. Fuentes și D.E. McArthur, "An Approach to multiple levels of detail generation from Digital Terrain Elevation Data using Wavelet Transforms", Proceedings of the 7<sup>th</sup> International Training Equipment Conference ITEC'96, pp. 255-262, April 1996, The Hague, The Netherlands.
- [Dog88] D. Dogaru, *Elemente de grafică 3D*, Ed. Științifică și Enciclopedică, București, 1988.
- [Drag57] M. Drăganu, *Introducere matematică în fizica teoretică modernă*, Editura Tehnică, 1957.
- [Fang95] Tsung-Pao Fang și Les A. Piegl, "Delaunay Triangulation in Three Dimensions", IEEE Computer Graphics and Applications, Vol. 15, Nr. 5, September 1995, pp. 62-69.
- [Earn93] R.A. Earnshaw, M.A. Gigante și H. Jones, *Virtual Reality Systems*, Academic Press, London, 1993.
- [Earn95] R.A. Earnshaw, J.A. Vince și H. Jones, *Virtual Reality Applications*, Academic Press, London, 1995.
- [Earn97] R.A. Earnshaw, "3D Multimedia on the Information Superhighway", IEEE Computer Graphics and Applications, Vol. 17, No. 2, March/April 1997.
- [Fjall93] Per-Olof Fjallstrom, "Evaluation of a Delaunay-based method for surface approximation", Computer-Aided Design, Vol. 25, Nr. 11, November 1993, pp. 711-719.
- [Foley90] J.D. Foley și A. Van Dam, *Computer Graphics: Principles and Practices*, 2<sup>nd</sup> Edn, Addison-Wesley, Reading, Massachusetts, 1990.
- [Ger97] N. Gershon și S.G. Eick, "Information Visualization", IEEE Computer Graphics and Applications, Vol. 17, No. 4, July/August 1997.
- [Gig93] M.A. Gigante, "Virtual Reality: Definitions, History and Applications", R.A. Earnshaw (ed), *Virtual Reality Systems*, Academic Press, London, 1993.
- [Gold94] B. Goldiez și Karen Williams, "Developing Simulator Networks", Proceedings of the International Training Equipment Conference ITEC'94, April 1994, The Hague, The Netherlands.
- [Gour71] H. Gouraud, "Continuous Shading of Curved Surfaces", IEEE Trans. on Computers, C-20(6), June 1971, pp. 623-629.
- [Grim93] M.A. Grimsdale, "SuperVision - A Parallel Architecture for Virtual Reality", R.A. Earnshaw (ed), *Virtual Reality Systems*, Academic Press, London, 1993.
- [Gross96] M.H. Gross, O.G. Staadt și R. Gatti, "Efficient Triangular Surface Approximation using Wavelets and Quadtree Data Structures", IEEE Transaction on Visualization and Computer Graphics, Vol. 2 No. 2, June 1996.
- [Hua75] T.S. Huang, *Picture Processing and Digital Filtering*, Springer-Verlag, Berlin, 1975.
- [Iacob96] A. Iacob, C. Bucur, C. Englert și Felicia Ionescu, "Romanian Approach, Achievements and Experience in Airline Pilots' Low Cost Training", in Proceedings of the East European Civil Aviation Training Conference, pp. 73-76, December 1996, Prague, Czech Republic.
- [Iacob97] A. Iacob și Felicia Ionescu, "Dynamics Model vs Visual Database Interaction in a Tank Driver Simulator", Proceedings of the 8<sup>th</sup> International Training and Education Conference ITEC'97, pp. 71-75, April 1997, Lausanne, Switzerland.
- [Ion96a] Felicia Ionescu, "Baze de date grafice orientate pe obiecte", *Revista Română de Informatică și Automatică*, vol. 6 (1996), pp. 37-43.



- [Ion96b] Felicia Ionescu, "Partiționarea Dinamică a Spațiului de Traversare a Bazelor de Date Grafice", *Revista Română de Informatică și Automatică*, vol.6 (1996), pp. 67-73.
- [Ion96c] Felicia Ionescu, "Scalability of Real-Time Parallel Image Synthesis Systems", *Revue Roumaine des Sciences Techniques. - Électrotechnique et Énergétique*, vol. 41 (1996), pp. 501-511.
- [Ion96d] Felicia Ionescu și Ș. Alexa, "PC-Based Software System for Visual Data Base Creation", *Proceedings of the 7<sup>th</sup> International Training Equipment Conference ITEC'96*, pp. 272-278, April 1996, The Hague, The Netherlands.
- [Ion96f] Felicia Ionescu, "Mapping Image Rendering Operations onto Parallel Processors", in *Proceedings of the International Semiconductor Conference CAS'96*, pp. 167-170, October 1996, Sinaia, Romania.
- [Ion96g] Felicia Ionescu, "Deadlock-Free Partial Multinode Broadcast in Image Rendering Parallel Networks", in *Proceedings of the International Symposium Communications'96*, pp. 147-152, November 1996, Bucharest, Romania.
- [Ion97a] Felicia Ionescu, "Controlul dinamic al granularității proceselor paralele de sinteză de imagine", *Revista Română de Informatică și Automatică*, vol. 7, (1997), pp. 5-11.
- [Ion97b] Felicia Ionescu, "Scheduling Real-Time Image Synthesis Operations in Multiprocessor Systems", *Studies in Informatics and Control*, vol. 6 (1997), pp. 173-180.
- [Ion97c] Felicia Ionescu, "A Solution for Multiple Levels of Detail Terrain Generation", *Proceedings of the 8<sup>th</sup> International Training and Education Conference ITEC'97*, pp. 23-28, April 1997, Lausanne, Switzerland.
- [Ion97d] Felicia Ionescu, "Codesign of a Parallel Architecture for Visual Systems", in *Proceedings of the 11<sup>th</sup> International Conference on Control Systems and Computer Science CSCS-11*, pp. 242-245, May 1997, București, Romania.
- [Ion98a] Felicia Ionescu, C. Gheorghiu, C. Englert, E. Popa și Iulia Suci, "Optimization of the execution time in a distributed system implementing a full flight simulator", in *Proceedings of the 9<sup>th</sup> International Training and Education Conference ITEC'98*, pp. 509-514, April 1998, Lausanne, Switzerland.
- [Ion98b] Felicia Ionescu, C. Gheorghiu, C. Englert și C. Stratulat, "Communications Protocols in Distributed Synthetic Environments", in *Proceedings of the International Conference Communications'98*, pp. 507-512, November 1998, Bucharest, Romania.
- [Ion98c] Felicia Ionescu, "Modelarea multirezoluție a formelor de teren", *Revista Română de Informatică și Automatică*, vol. 8, (1998), pp. 45 - 52.
- [Ion99a] Felicia Ionescu, *Principiile Calculului Paralel*, Editura Tehnică, București 1999.
- [Ion99b] Felicia Ionescu și M. Ionescu, "Pipeline Processing of Shared Graphs in Multiprocessor Systems", *International Conference on Parallel and Distributed Processing Techniques and Applications*, pp 1958-1963, June 1999, Las Vegas, Nevada, USA.
- [Ion00a] Felicia Ionescu, "Application-level Virtual Memory Management in Real-Time Multiprocessor Systems", in *Proceedings of the 14<sup>th</sup> Annual Symposium for Applied Computing ACM SAC-2000*, March 2000, Como, Italy.
- [Ion00b] Felicia Ionescu și A. Jalbă, "Data Parallelism vs. Control Parallelism in Synthetic Environments Visualization", in *Proceedings of the 11<sup>th</sup> International Training and Education Conference ITEC'2000*, April 2000, The Hague, The Netherlands.
- [Kal89] Y.E. Kalay, *Modeling Objects and Environment*, Wiley&Sons, New York, 1989.
- [Ken89] R.S. Kennedy, M.G. Lilienthal, K.S. Berbaum și M.E. McCauley, "Simulator sickness in U.S Navy Flight Simulators", *Aviation, Space and Environmental Medicine*, 1989.
- [Lear97] Anne C. Lear, "Virtual Reality provides Real Therapy", *IEEE Computer Graphics and Applications*, Vol. 17, No. 4, July-August 1997.
- [Lou97] M. Louka, *An Introduction to Virtual Reality*, Ostfold Colledge, 1997.
- [Man95] C. Manetta și R. Blade, "Glossary of Virtual Reality Terminology", *International Journal of Virtual Reality*, Vol.1, Nr. 2, 1995.
- [Mat99] S.N. Matsuba și B. Roehl, "The making of VRML Dream", *IEEE Computer Graphics and Applications*, March/April, Vol. 19, Nr. 2, 1999.
- [Mold96] Florica Moldoveanu, Zoea Racoviță, Ș. Petrescu, G. Hera și M. Zaharia, *Grafica pe Calculator*, Editura Teora, 1996.



- [Nad99] D.R. Nadeau, "Tutorial: Building Virtual Worlds with VRML", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.
- [New72] M.E. Newell, R.G. Newell și T.L. Sancha, "A new Approach to the shaded picture problem", Proc. ACM National Conference, pp. 443-450, 1972.
- [New81] W.M. Newman și R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New Zork, 1981.
- [Nor98] M.M North, S.M. North, și J.R. Coble, "Virtual Reality Therapy: An effective Treatment for the Fear of Public Speaking", Int'l J. of Virtual Reality, Vol. 3, No. 2, 1998.
- [Pap77] A. Papoulis, *Signal Analysis*, McGraw-Hill, New York, 1977.
- [Par95] R. Pares i Burgues și N. Pares i Burgues, "GaleriaVirtual: a platform for non-limitation in comtemporary art?", R.A.Earnshaw (ed), Virtual Reality Applications, Academic Press, London 1995.
- [Prep85] F.P. Preparata și M.I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New-York, 1985.
- [Phong75] B. Phong, "Illumination for computer-generated pictures", Comm. ACM, Vol. 18, Nr. 6, 1975, pp. 311-317.
- [Reg95] E.C. Regan, "Some human factors issues in immersive virtual reality: fact and speculation", R.A.Earnshaw (ed), Virtual Reality Applications, Academic Press, London 1995.
- [Rob93] W. Robunett și J.P. Rolland, "A Computational Model for Stereoscopic Optics of a Head-Mounted Display", R.A. Earnshaw (ed), Virtual Reality Systems, Acad. Press, London, 1993.
- [Row93] T.W. Rowley, "Virtual Reality Products", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993.
- [Sat92] R. Satava, "Virtual Reality Surgical Simulator", Proceeding Medicine Meets Virtual Reality Conference, San Diego, CA, 1993.
- [Sch98] M. Schulz, T. Reuding și T. Ertl, "Analyzing Engineering Simulations in a Virtual Environment", IEEE Computer Graphics and Applications, Vol. 18, No. 6, 1998.
- [SGI96] Silicon Graphics, *Onyx2 Technical Report*, Mountain View, CA, 1996.
- [Slat99] M. Slater, D.P. Pertaub și A. Steed, "Public Speaking in Virtual Reality", IEEE Computer Graphics and Applications, March-April 1999.
- [Suth74] I.E. Sutherland și G.W. Hodgman, "Reentrant polygon clipping", Comm. ACM, Vol. 17, No. 1, pp. 32-42, 1974.
- [Rour93] Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, Boston, Massachusetts, 1993.
- [Sat95] R.M. Satava, "Virtual Reality for the phisicians of the 21<sup>st</sup> century", R.A.Earnshaw (ed), Virtual Reality Applications, Academic Press, London 1995.
- [Sla99] M. Slater, D.P. Pertaub și A. Steed, "Public Speaking in Virtual Reality", IEEE Computer Graphics and Applications, Vol. 19, No. 2, March/April 1999.
- [Sher92] B. Sherman și P. Judkins, *Glimpses of Heaven, Vision of Hell: Virtual Reality and its implications*, Hodder and Stoughton, London, 1992.
- [Vin93] J. Vince, "Virtual Reality Techniques in Flight Simulation", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London, 1993.
- [Vin97] J. Vince, *Virtual Reality Systems*, Addison-Wsley, New York, 1997.
- [Watt95] A. Watt, *3D Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1995.
- [Wei77] K. Weiler și P. Atherton, "Hidden surface removal using polygon area sorting", Computer Graphics, Vol. 11, No. 2, 1977, pp. 214-222.
- [Wha93] L.J. Whalley, "Ethical Issues in VR Treatment of Mental Disorders", R.A. Earnshaw (ed), Virtual Reality Systems, Academic Press, London 1993.
- [Woo97] M. Woo, Jackie Neider și T. Davis, *OpenGL Programming Guide*, OpenGL Architecture Review Board, Addison-Wesley Developers Press, Reading, MA 1997.
- [Xia97] Julie C. Xia, J. El-Sana și A. Varshney, "Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models", IEEE Transactions on Visualization and Graphics, Vol. 3, No. 2, April-June 1997.
- [You98] P. Young, *3D Graphics and Virtual Reality*, online <http://pc-vrg.dur.ac.uk/cg-notes/>



# INDEX BILINGV DE TERMENI

Accelerator grafic	14	buffer de imagine (de culoare)	133
<i>graphic accelerator</i>		<i>color buffer</i>	
algoritm de triangularizare	40	buffer șablon	135
incremental		<i>stencil buffer</i>	
<i>incremental triangulation alg.</i>			
algoritmul de baleiere pe linii	114	Căști binaurale	18
<i>scan-line conversion</i>		<i>binaural phones</i>	
arbore cuaternar	48	ceață	173
<i>quadtree</i>		<i>fog</i>	
arbore octal	48	centru de proiecție	83
<i>octree</i>		<i>centre of projection</i>	
aliasing	217	ciclu de vârfuri	26
<i>aliasing</i>		<i>vertices cycle</i>	
animație	299	coduri de vizibilitate	101
<i>animation</i>		<i>visibility codes</i>	
anti-aliasing	217	compunerea transformărilor	57
<i>anti-aliasing</i>		<i>transformation concatenation</i>	
A-buffer	224	construcție corectă a poliedrelor	29
<i>A-buffer</i>		<i>well-formedness</i>	
		context de afișare	133
Bază de date grafice	264	<i>device context</i>	
<i>graphic data-base</i>		context de redare	133
biblioteci grafice	15	<i>rendering context</i>	
<i>graphics library</i>		continuitate pozițională	204
B-spline neuniforme raționale	206	<i>positional continuity</i>	
<i>Non-uniform rational B-spline</i>		continuitate tangențială	204
<i>curves-NURBS</i>		<i>tangential continuity</i>	
buffer de acumulare	135	coordonate de texturare	240
<i>accumulation buffer</i>		<i>texture coordinates</i>	
buffer de adâncime	119	corecție gamma	24
<i>Z-buffer (depth buffer)</i>		<i>gamma correction</i>	
buffer de cadru	133	cromicitate (saturație)	23
<i>frame buffer</i>		<i>chroma (saturation)</i>	



curbe Bézier	195	fragment de suprafață	223
<i>Bézier curves</i>		<i>surface fragment</i>	
curbe B-spline	197	frecvența de redare a imaginilor	10
<i>B-spline curves</i>		<i>update rate</i>	
Decupare	100	forfecare	73
<i>clipping</i>		<i>shear</i>	
deteția coliziunii	110	forța de reacție	19
<i>collision detection</i>		<i>force feedback</i>	
diagonală în poligon	28	funcția de reflexie bidirecțională	164
<i>polygon diagonal</i>		<i>bidirectional reflection function</i>	
diagrama Voronoi	37	funcții de amestec	46
<i>Voronoi diagraphs</i>		<i>blending functions</i>	
direcția de proiecție	83	Generarea volumelor prin baleiere	35
<i>direction of projection</i>		<i>volume sweeping</i>	
display montat pe cap	3	generator de imagine	10
<i>head-mounted display(HMD)</i>		<i>image generator</i>	
disp. de captare și urmărire a poziției	8	girație	53
<i>tracker</i>		<i>yaw, heading</i>	
Eliminarea fețelor orientate invers	123	graful scenei virtuale	263
<i>backface elimination</i>		<i>virtual scene graph</i>	
eliminarea obiectelor invizibile	108	Harta digitală a terenului	41
<i>culling</i>		<i>Digital Terrain Elevation Data</i>	
eliminarea suprafețelor ascunse	116	Imersivitate	4
<i>hidden surface removal</i>		<i>immersivity</i>	
estomparea imaginilor în mișcare	231	Juxtapunerea imaginilor	16
<i>motion blur</i>		<i>edge-blending</i>	
eșantionarea imaginii	218	Legea lui Moebius	30
<i>image sampling</i>		<i>Moebius law</i>	
eșantionarea stocastică	227	limbajul VRML	289
<i>stochastic sampling</i>		<i>VRML language</i>	
Față orientată direct	141	limita Nyquist	219
<i>frontface</i>		<i>Nyquist limit</i>	
față orientată invers	141	linie poligonală	27
<i>backface</i>		<i>polygonal line</i>	
față poligonală	29	link-ul unui vârf	29
<i>polygonal face</i>		<i>vertex link</i>	
fereastra de vizualizare	86	listă de display	119
<i>view plane window</i>		<i>display list</i>	
filtrare de mărire	255	lumina absorbită	164
<i>magnification</i>		<i>absorbed light</i>	
filtrare de micșorare	255	lumină ambientală	168
<i>minification</i>		<i>ambient light</i>	
filtre variabile în spațiu	243	lumina împrăștiată	164
<i>space-variant filters</i>		<i>scattered light</i>	
filtru bidimensional	219		
<i>two-dimensional filter</i>			



- |                                      |     |                                       |     |
|--------------------------------------|-----|---------------------------------------|-----|
| lumină incidentă                     | 164 | Obiecte deformabile                   | 25  |
| <i>incident light</i>                |     | <i>deformable objects</i>             |     |
| lumina reflectată                    | 164 | obiecte solide                        | 25  |
| <i>reflected light</i>               |     | <i>solid objects</i>                  |     |
| lumină transmisă                     | 164 | orientare consistentă a fețelor       | 30  |
| <i>transmitted light</i>             |     | <i>surface consistent orientation</i> |     |
| Matrice de transformare curentă      | 145 | Păclă                                 | 173 |
| <i>current transformation matrix</i> |     | <i>haze</i>                           |     |
| mănușă de date                       | 19  | petic de suprafață                    | 45  |
| <i>data glove</i>                    |     | <i>patch</i>                          |     |
| mediu sintetic                       | 1   | plan de proiecție (de vizualizare)    | 82  |
| <i>synthetic environment</i>         |     | <i>view plane</i>                     |     |
| mediu virtual                        | 1   | plan de vizibilitate apropiat         | 86  |
| <i>virtual environment</i>           |     | <i>near plane</i>                     |     |
| metoda cercului vid                  | 39  | plan de vizibilitate depărtat         | 86  |
| <i>empty circle method</i>           |     | <i>far plane</i>                      |     |
| modelare prin divizare spațială      | 48  | poliedru                              | 29  |
| <i>space subdivision modeling</i>    |     | <i>polyedron</i>                      |     |
| modelarea obiectelor                 | 25  | poliedru de control                   | 202 |
| <i>object modeling</i>               |     | <i>control polyedron</i>              |     |
| modelare poligonală                  | 26  | poligon                               | 26  |
| <i>polygonal modeling</i>            |     | <i>polygon</i>                        |     |
| model de iluminare                   | 163 | poligon de control                    | 195 |
| <i>illumination model</i>            |     | <i>control polygon</i>                |     |
| model de reflexie                    | 163 | pre-image pixel                       | 241 |
| <i>reflexion model</i>               |     | <i>pixel pre-image</i>                |     |
| modelare prin compunerea obiectelor  | 46  | prefiltrarea imaginii                 | 221 |
| <i>Constructive Solid Geometry</i>   |     | <i>image prefiltering</i>             |     |
| modelare prin rețele de petice       | 45  | primitive geometrice                  | 135 |
| <i>patch nets modelling</i>          |     | <i>geometric primitives</i>           |     |
| modelul HSV                          | 22  | proiecție ortografică                 | 83  |
| <i>HSV model</i>                     |     | <i>orthographic projection</i>        |     |
| modelul RGB                          | 21  | proiecție paralelă                    | 83  |
| <i>RGB model</i>                     |     | <i>parallel projection</i>            |     |
| muchie poligonală                    | 27  | proiecție perspectivă                 | 83  |
| <i>edge</i>                          |     | <i>perspective projection</i>         |     |
| muzeu virtual                        | 308 | proprietatea cercului vid             | 39  |
| <i>virtual museum</i>                |     | <i>empty circle property</i>          |     |
| Nivele de detaliu                    | 34  | puncte de control                     | 195 |
| <i>levels of detail (LOD)</i>        |     | <i>control points</i>                 |     |
| noduri                               | 200 | Ray tracing                           | 47  |
| <i>knots</i>                         |     | <i>ray tracing</i>                    |     |
| normala în vârf                      | 170 | rău de mișcare                        | 307 |
| <i>vertex normal</i>                 |     | <i>motion sickness</i>                |     |
| normala la suprafață                 | 124 | rău de simulator                      | 307 |
| <i>surface normal</i>                |     | <i>simulator sickness</i>             |     |
| nuanță                               | 23  |                                       |     |
| <i>hue</i>                           |     |                                       |     |

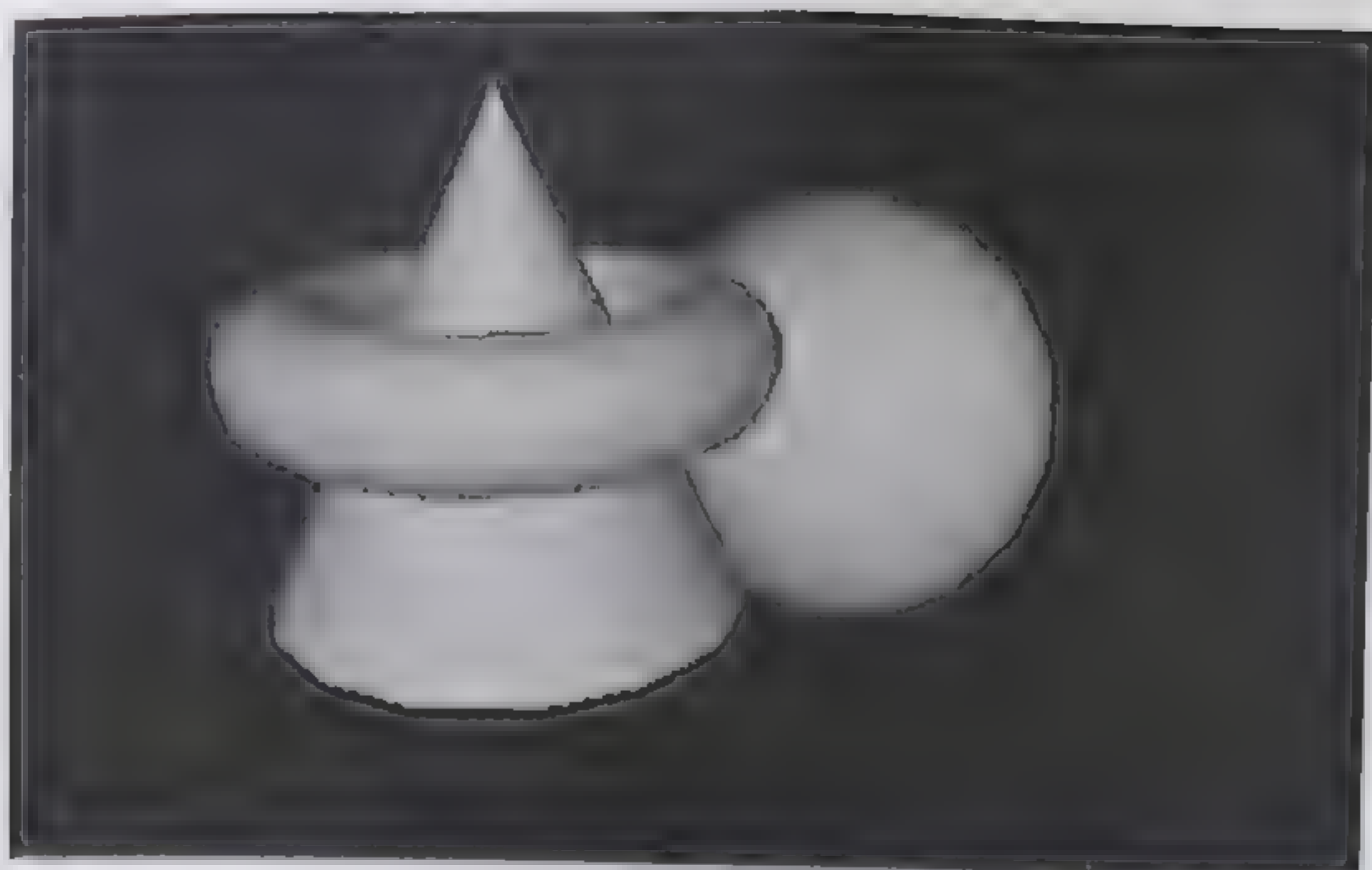


reacție haptică	19	sistem de referință de modelare	75
<i>haptic feedback</i>		<i>modeling coordinate system</i>	
realitate artificială	1	sistem de referință normalizat	85
<i>artificial reality</i>		<i>normalized coordinates</i>	
realitate virtuală desktop	3	sistem de referință de observare	76
<i>desktop VR</i>		<i>viewing coordinate system</i>	
realitate îmbogățită	3	sistem de referință geocentric	41
<i>augmented reality</i>		<i>geocentric reference</i>	
realitate virtuală	1	sistem de proiecție Mercator	41
<i>virtual reality</i>		<i>Universal Transversal Mercator</i>	
realitate virtuală imersivă	3	<i>(UTM)</i>	
<i>immersive VR</i>		sistem de vizualizare	75
realitate virtuală proiectivă	3	<i>viewing system</i>	
<i>projected VR</i>		sistem de vizualizare standard	92
reflexie difuză	166	<i>standard viewing system</i>	
<i>diffuse reflection</i>		spațiul texturii	236
reflexie speculară	167	<i>texture space</i>	
<i>specular reflexion</i>		solide extrudate	35
rezoluția imaginii	11	<i>extruded solids</i>	
<i>image resolution</i>		stiva matricelor de modelare-	147
redare volumetrică	50	vizualizare	
<i>volume rendering</i>		<i>modelview matrix stack</i>	
reprezentare "plină"	33	stiva matricelor de proiecție	148
<i>filled representation</i>		<i>projection matrix stack</i>	
reprezentare "cadru de sârmă"	33	stiva matricelor de texturare	148
<i>wireframe representation</i>		<i>texture matrix stack</i>	
rotație	52	stivele matricelor de transformare	147
<i>rotation</i>		<i>matrix stacks</i>	
ruliu	53	supraeșantionare (postfiltrare)	225
<i>roll</i>		<i>supersampling (postfiltering)</i>	
Scalare	52	suprafață (petic) Bézier	202
<i>scale</i>		<i>Bézier surface (patch)</i>	
scena virtuală	70	suprafață (petic) B-spline	205
<i>virtual scene</i>		<i>B-spline surface (patch)</i>	
senzația tactilă	19	suprafața de frontieră	26
<i>touch feedback</i>		<i>boundary representation</i>	
sistem de coordonate omogene	54	<i>(B-rep)</i>	
<i>homogeneous coordinates</i>		sursă de lumină	175
sistem de coordonate drept	20	<i>light source</i>	
<i>right-handed coordinates</i>		Tangaj	53
sistem de coordonate stâng	20	<i>pitch</i>	
<i>left-handed coordinates</i>		teatru virtual	306
sistem de referință universal	20	<i>virtual theatre</i>	
<i>world coordinate system</i>		model de umbrire	164
sistem geodezic universal	41	<i>shading model</i>	
<i>World Geodetic System</i>		tehnica chirurgicală cu invaziune	305
<i>(WGS)</i>		minimă	
sistem de referință ecran 3D	97	<i>minimal invasion technique</i>	
<i>3D screen reference</i>			



teleprezență	3	translație	51
<i>telepresence</i>		<i>translation</i>	
texel	236	transparență	176
<i>texel</i>		<i>tranparency</i>	
textură bidimensională	236	traversarea grafului scenei	266
<i>two-dimensional texture</i>		<i>scene graph traversal</i>	
textură mip-map	245	triangularizare	27
<i>mip-map texture</i>		<i>triangulation</i>	
textură tridimensională	241	triangularizarea Delaunay în plan	38
<i>three-dimensional texture</i>		<i>planar Delaunay triangulation</i>	
texturare	235	triangularizarea Delaunay în spațiu	40
<i>texture mapping</i>		<i>spatial Delaunay triangulation</i>	
transformări geometrice în spațiu	51	trunchi de piramidă de vizualizare	85
<i>three-dimensional transformation</i>		<i>viewing frustum</i>	
transformări geometrice în plan	74	Umbrire constantă	169
<i>two-dimensional transformations</i>		<i>flat shadow</i>	
transformări geometrice primitive	51	umbrire incrementală Gouraud	169
<i>primitive geometric transf.</i>		<i>smooth Gouraud shading</i>	
transformări inverse	61	umbrire incrementală Phong	172
<i>inverse transformations</i>		<i>smooth Phong shading</i>	
transformarea de modelare	70	unghiul de vizibilitate	17
<i>modeling transformation</i>		<i>field of view</i>	
transformarea de normalizare	88	Vârf al poligonului	27
<i>normalization transformation</i>		<i>polygon vertex</i>	
transformarea de observare	77	vedere stereoscopică	18
<i>view transformation</i>		<i>stereoscopic view</i>	
transformarea de parametrizare	236	viteza de generare a imaginii	10
<i>parametrization</i>		<i>image generation speed</i>	
transformarea de proiecție	82	volum de delimitare	109
<i>projection transformation</i>		<i>bounding box</i>	
transformarea de rastru	111	volum de vizualizare	86
<i>rasterization</i>		<i>viewing volume</i>	
transformare de texturare	240	volum canonic	87
<i>texture mapping</i>		<i>canonical view volume</i>	
transformare sisteme de referință	63	voxel	48
<i>coordinate system transformation</i>		<i>voxel</i>	
transformata Fourier	218		
bidimensională			
<i>two-dimensional Fourier</i>			
<i>transform</i>			





a)



b)



c)

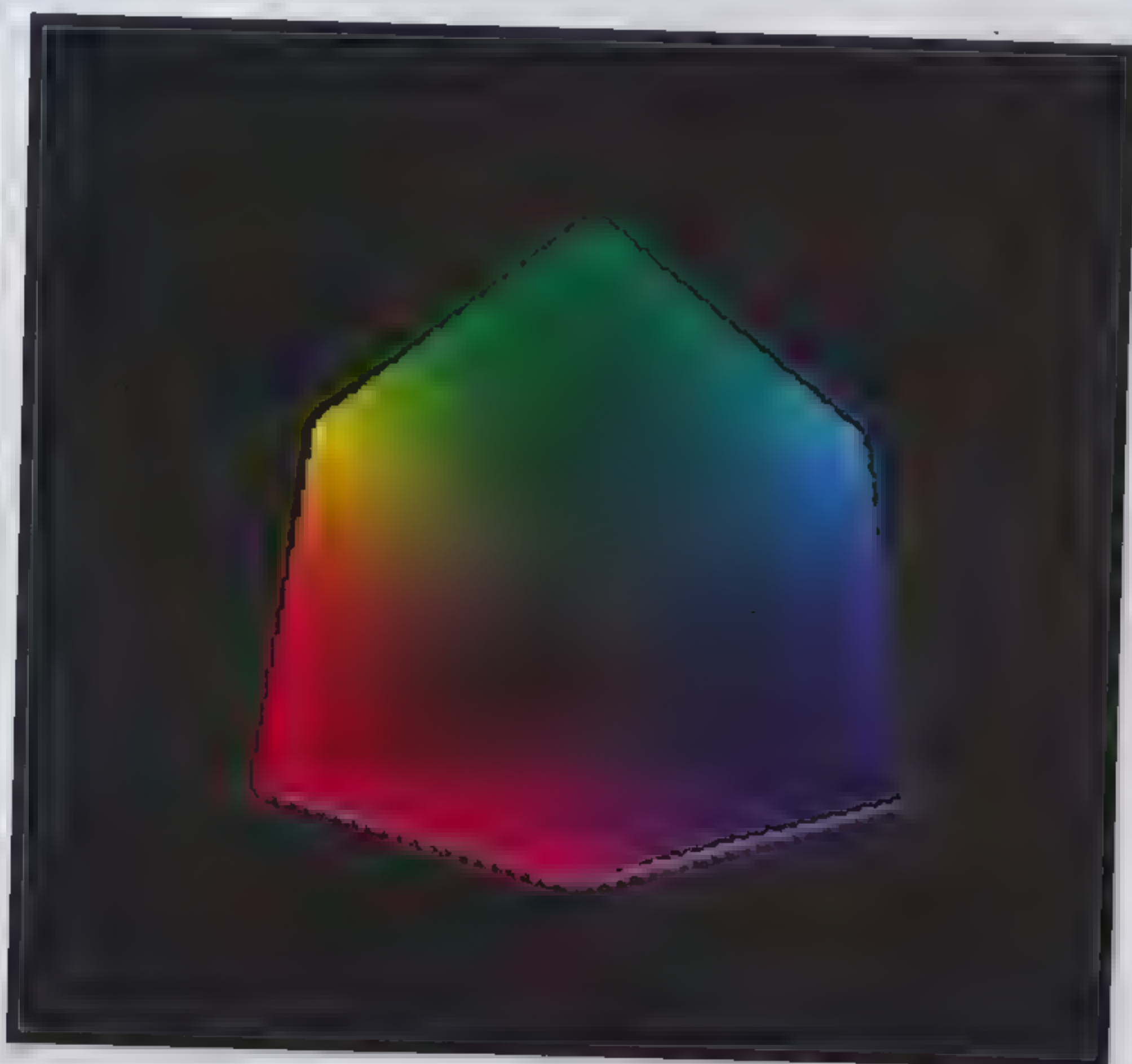


d)

**PLANȘA 1.** Obiecte din material cu reflectanță ambientală, difuză și speculară gri, și sursă de lumină colorată: **a)** obiectele iluminate de o sursă de lumină cu componente ambientală și difuză gri; **b)** obiectele iluminate de o sursă de lumină cu componenta ambientală gri și componenta difuză albastră; **c)** și **d)** obiectele iluminate de o sursă de lumină cu componenta ambientală gri, componenta difuză albastră și componenta speculară albă; în **c)** exponentul de reflexie speculară are valoare mică (20); în **d)** exponentul de reflexie speculară are valoare mare (80).



a)



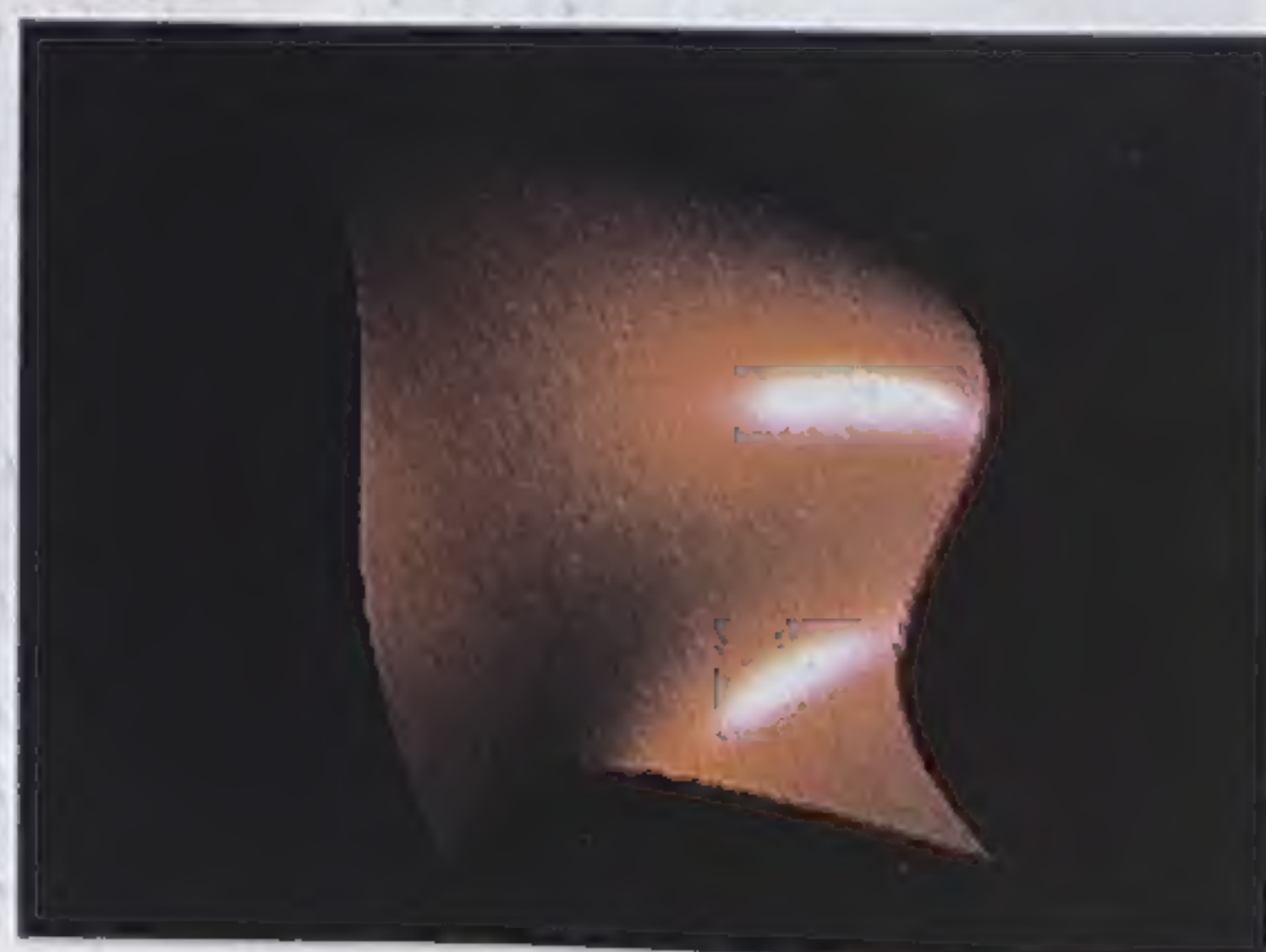
b)

**PLANȘA 2.** Cubul culorilor RGB: **a)** axele (Cyan, Magenta, Yellow - CMY); **b)** axele (Red, Green, Blue).



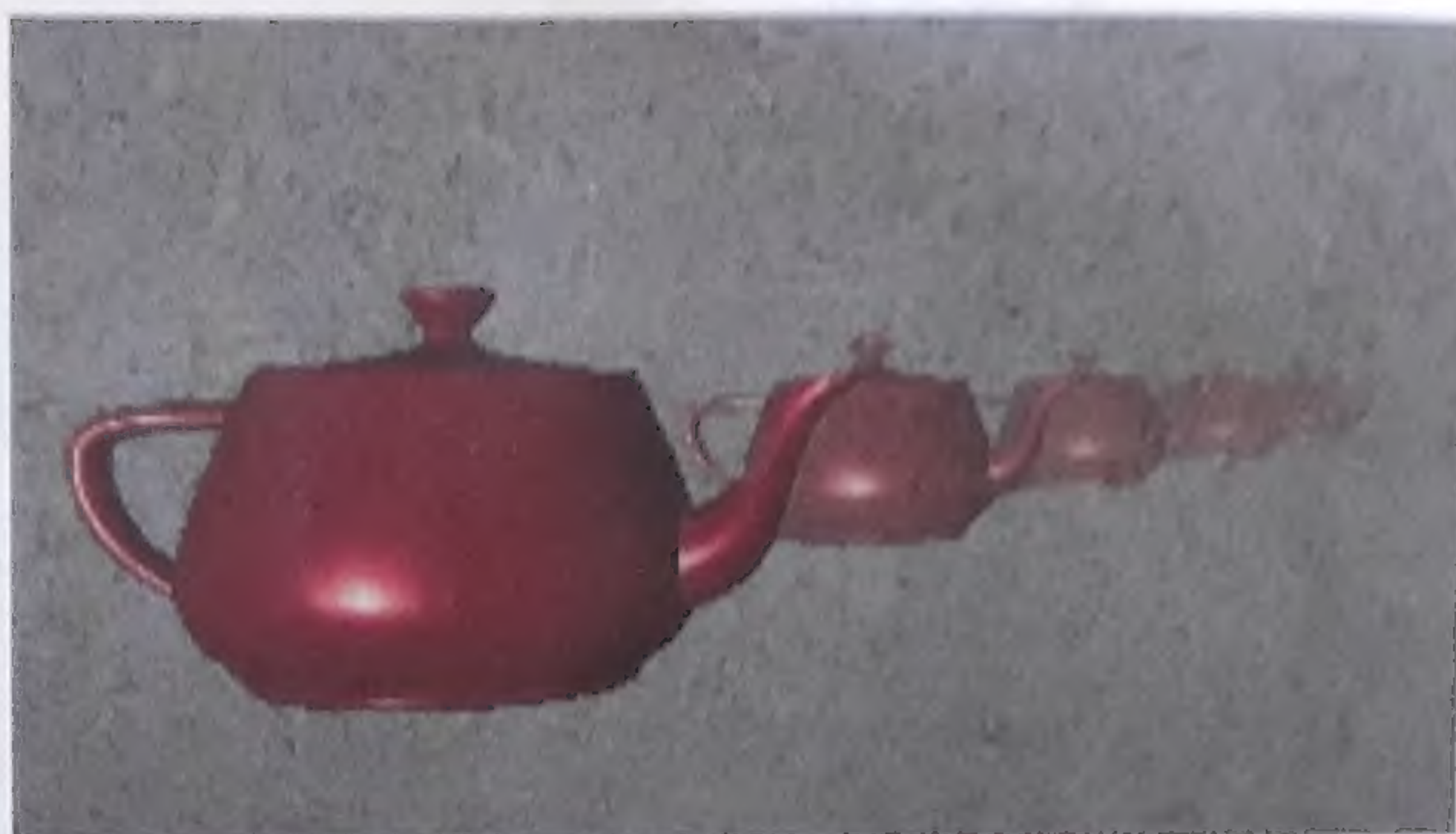


**PLANȘA 3.** Obiecte iluminate, cu umbrire Gouraud și diferite proprietăți de material care imită materiale reale. *Pe prima coloană* (de sus în jos): jad, obsidian, perlă, rubin, turcoaz. *A doua coloană*: bronz, crom, cupru, aur, argint. În a treia coloană sunt obiecte din materiale plastice de diferite culori: cian, verde, roșu, alb, galben. În coloana a patra sunt obiecte din cauciuc cu culori similare.



**PLANȘA 4.** Obiecte redată cu iluminare și umbrire. În partea stângă, sferă cu umbrire poligonală și umbrire Gouraud. În partea dreaptă, suprafață Bézier cu umbrire Gouraud.





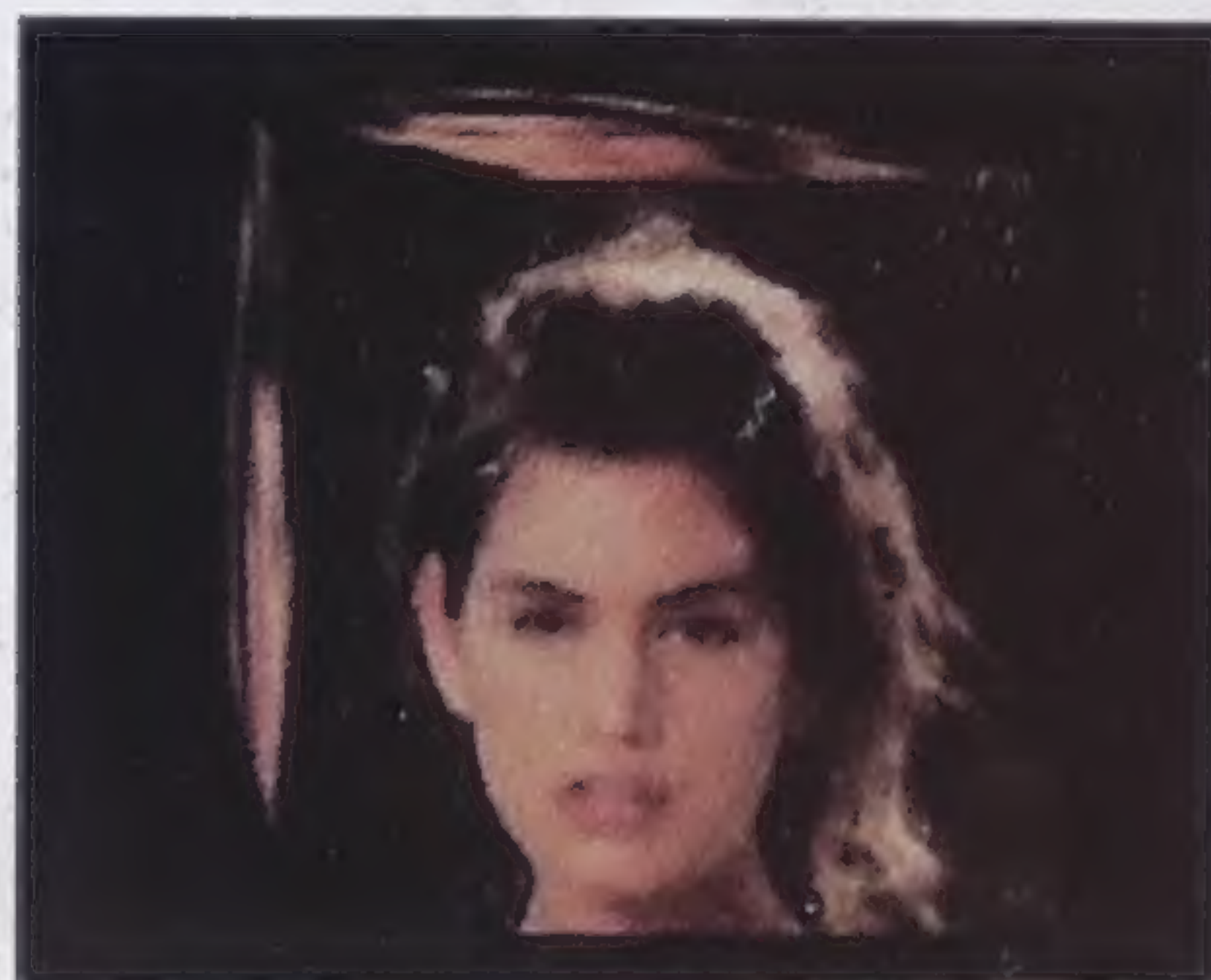
**PLANȘA 5.** Combinarea culorilor. În partea stângă, suprafețe transparente suprapuse; triunghiul de culoare cian este transparent și suprapus peste triunghiul de culoare galbenă. În partea dreaptă, simularea efectelor atmosferice (ceață).



**PLANȘA 6.** Obiecte cu iluminare și umbră. La stânga, obiectul este desenat cu aliasing. La dreapta, obiectul este desenat cu anti-aliasing folosind un buffer de acumulare.



a)

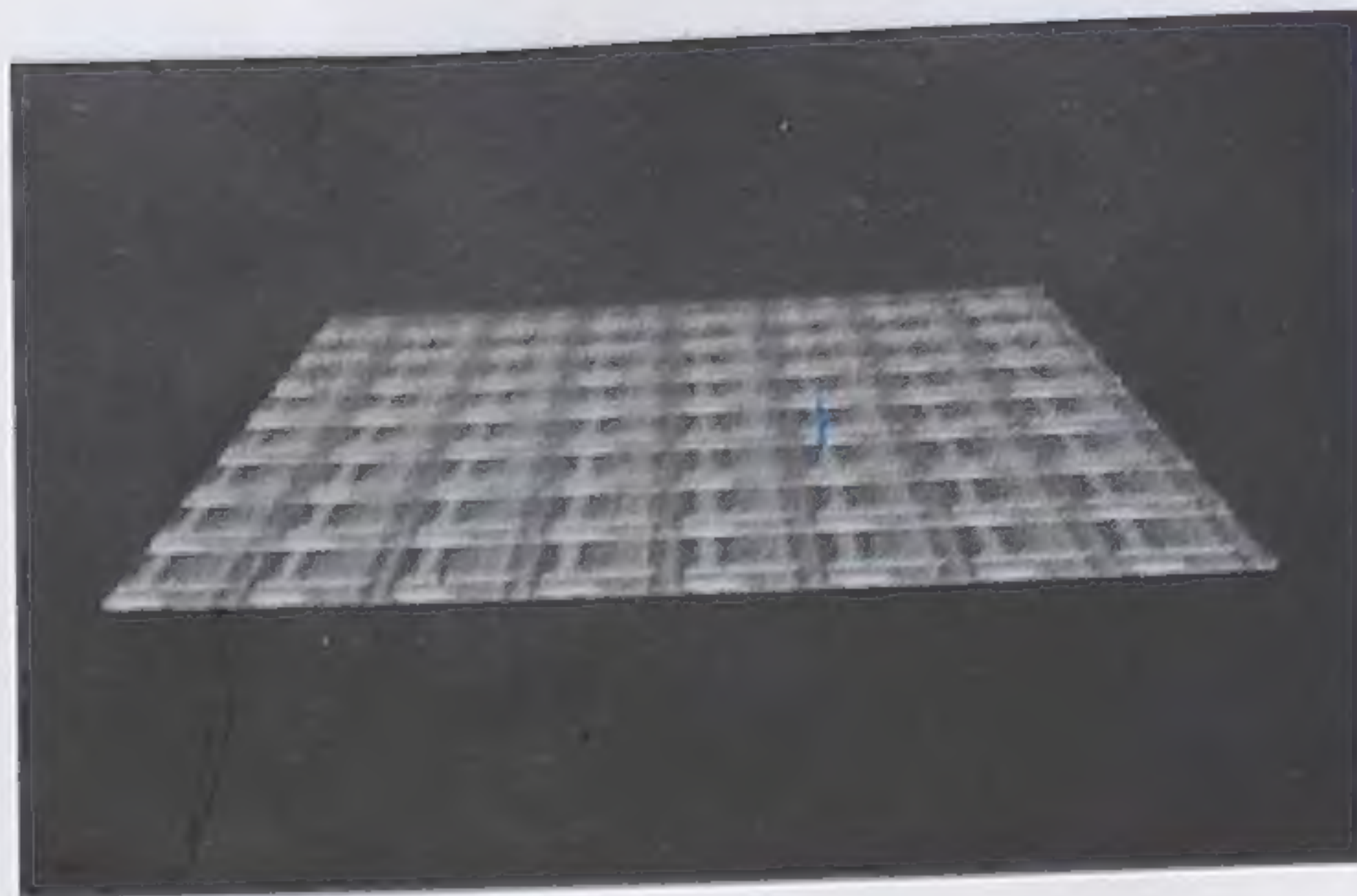
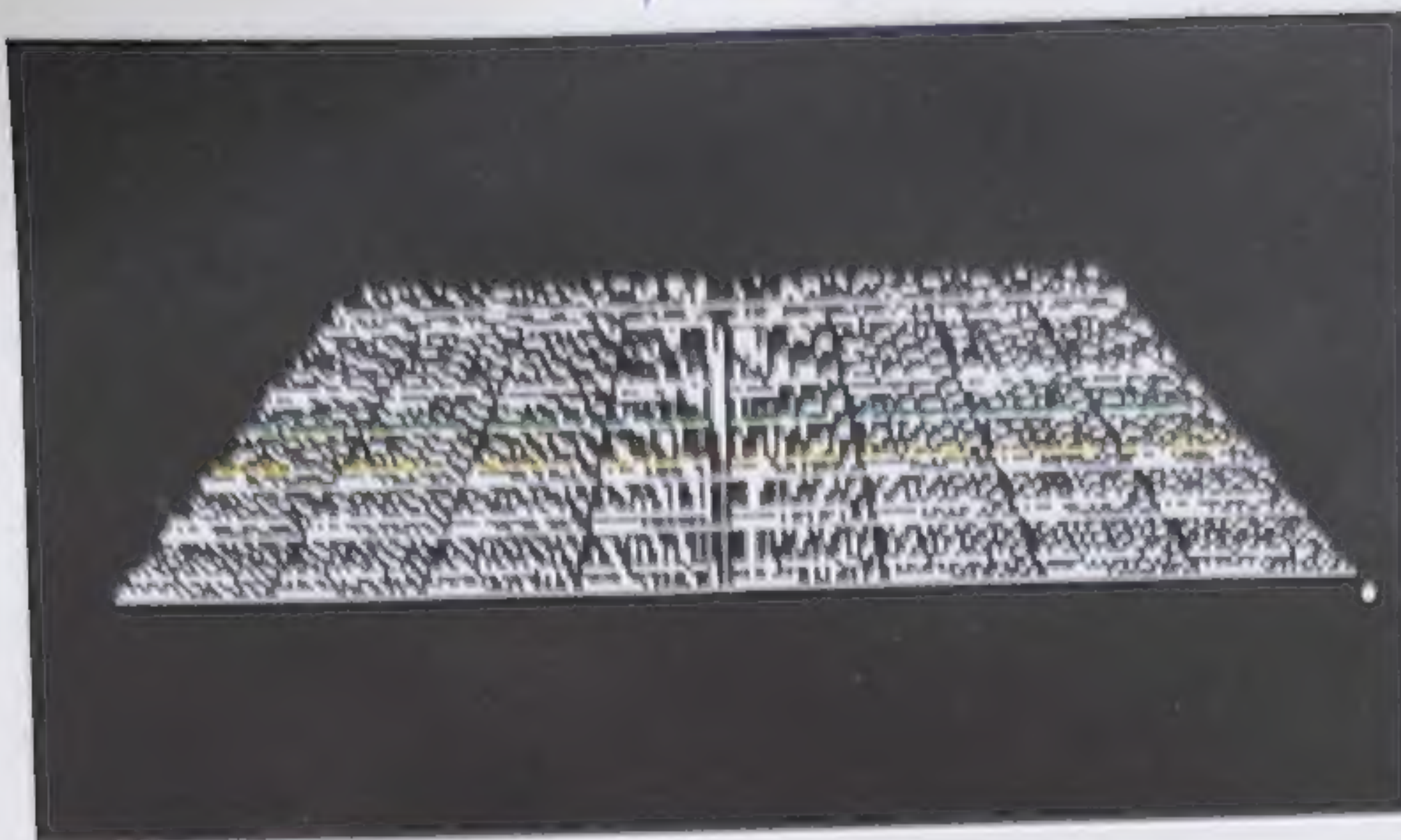


b)

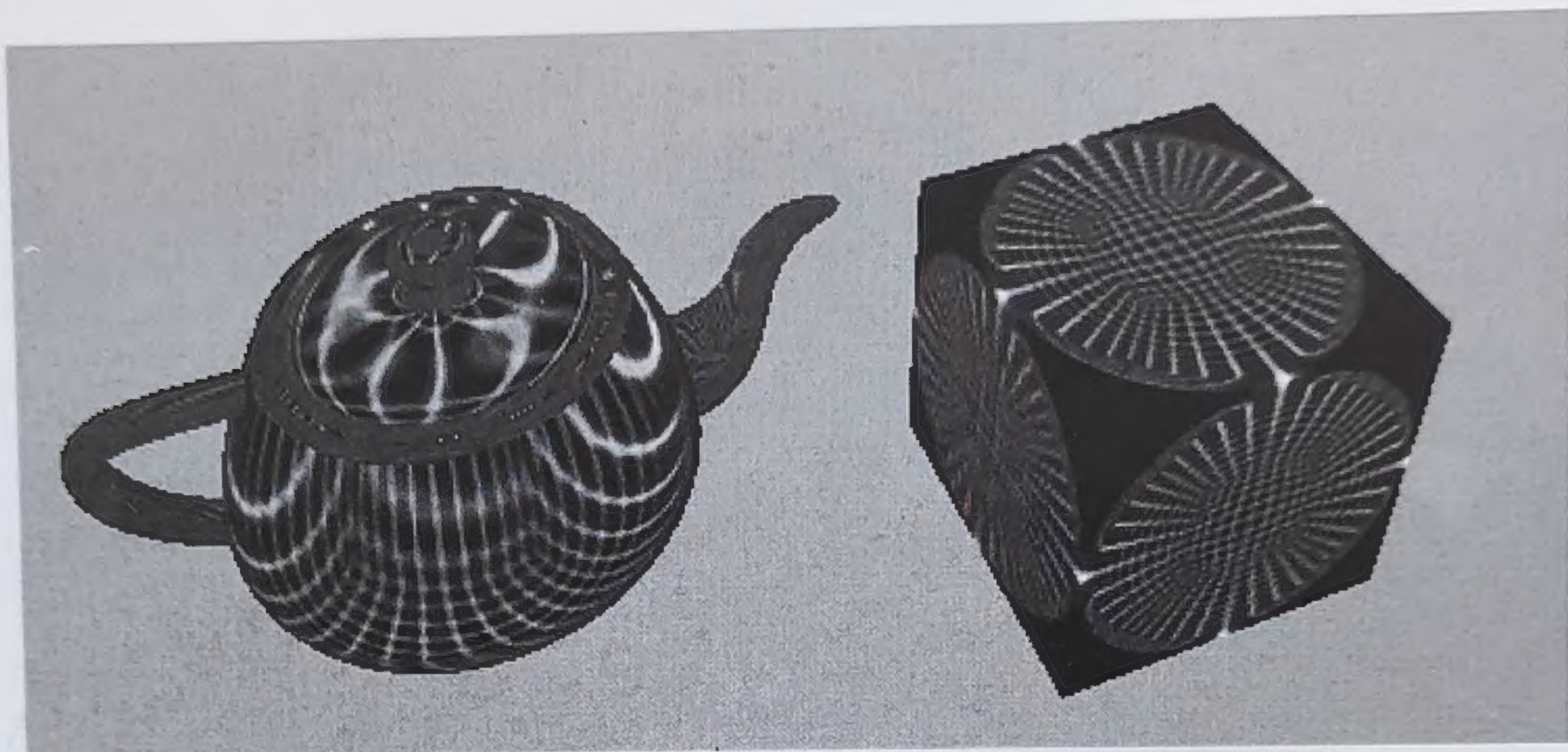
**PLANȘA 7.** Suprafețe ale obiectelor texturate prin aplicarea unei texturi bidimensionale:  
a) aplicația repetată a imaginii unei cărămizi;  
b) aplicația unei fotografii.



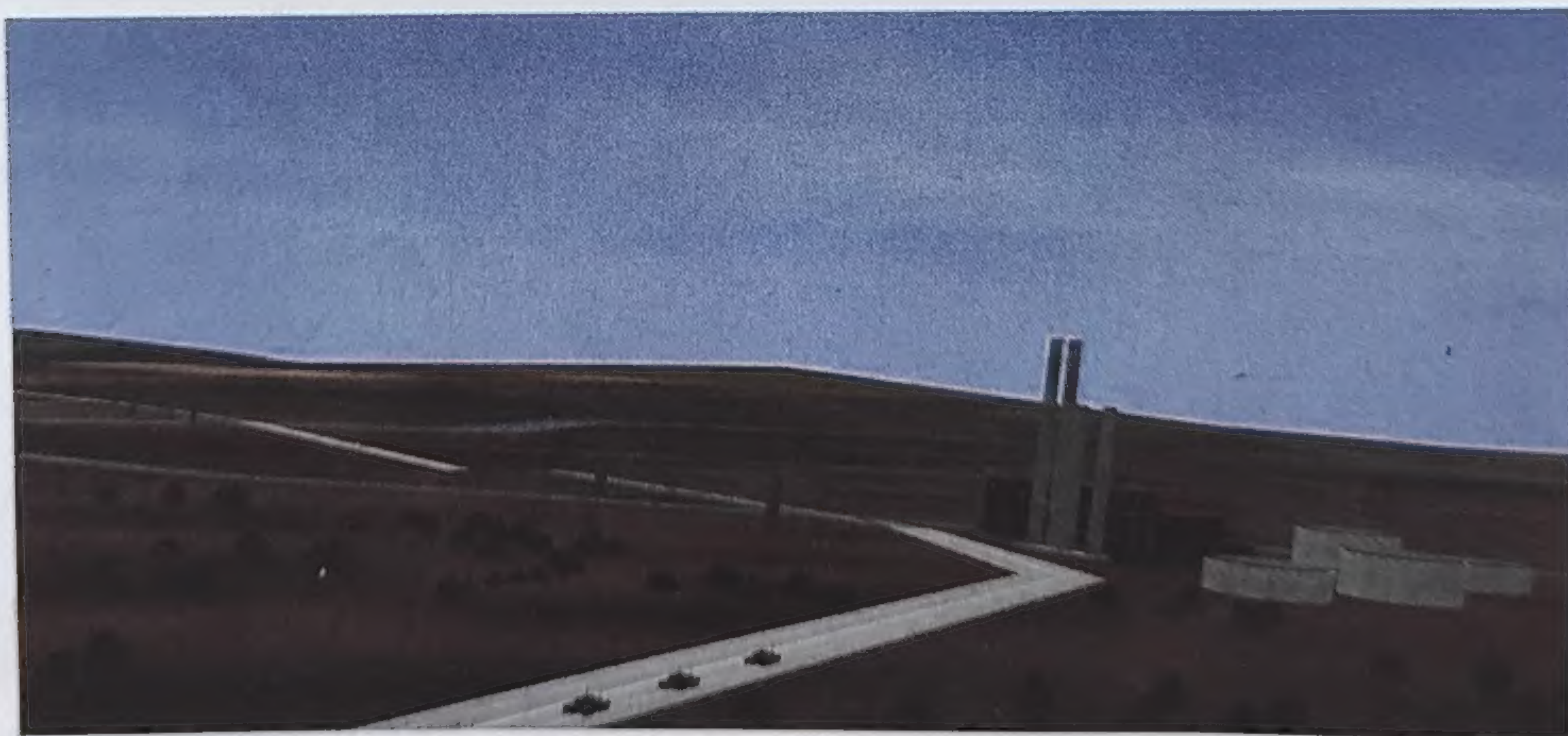
2754/2000



PLANȘA 8. a) Imagine cu textură nefiltrată. b) Imagine cu aceeași textură filtrată.



PLANȘA 9. Aplicația sferică a texturii pe obiectul „ceainic” și aplicația plană pe fiecare față a cubului.



PLANȘA 10. Imagine dintr-un simulator de zbor. Suprafețele sunt texturate, cu umbrire Gouraud și anti-aliasing.

B.C.U. „M. EMINESCU” IASI

50,000 lei



**Prof. univ. dr. ing. FELICIA IONESCU** este titular al disciplinelor *Sisteme Paralele și Distribuite*, *Grafica în realitatea virtuală*, *Programare obiect-orientată* în cadrul Catedrei de Electronică Aplicată și Ingineria Informației din Facultatea de Electronică și Telecomunicații, Universitatea "POLITEHNICA" București. În activitatea de cercetare pe care a desfășurat-o la *Institutul de Tehnică de Calcul București* și apoi la *Institutul de Simulatoare Simultec*, a coordonat activitatea de cercetare și proiectare a mai multor sisteme de generare a imaginii, integrate în simulatoare de antrenament pentru piloți.

Lucrarea **GRAFICA ÎN REALITATEA VIRTUALĂ** prezintă aspecte teoretice și tehnici de programare a aplicațiilor grafice, în special cele necesare în sistemele de realitate virtuală:

- Sisteme și aplicații ale realității virtuale
- Modelarea obiectelor și a scenelor virtuale
- Sisteme de vizualizare
- Modele de reflexie și iluminare
- Tehnici de creare a realismului vizual: umbrire, anti-aliasing, texturare
- Exemple ilustrative de programare a aplicațiilor grafice folosind biblioteca grafică OpenGL și limbajul VRML